

# **Evidence-based Software Engineering**

**Based on the publicly available data**

**Derek M. Jones**

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Software ecosystems . . . . .	2
1.1.1	The primary activities of software engineering . . . . .	4
1.2	Brief history of software engineering research . . . . .	5
1.2.1	Folklore . . . . .	7
1.2.2	The academic ecosystem . . . . .	8
1.3	Lessons learned from the analysis in this book . . . . .	10
1.4	Overview of contents . . . . .	10
1.4.1	Why use R? . . . . .	12
1.5	Terminology, concepts and notation . . . . .	13
<b>2</b>	<b>Human cognitive characteristics</b>	<b>15</b>
2.1	Introduction . . . . .	15
2.1.1	Modeling human cognition . . . . .	17
2.1.2	Embodied cognition . . . . .	18
2.1.3	Perfection is not cost-effective . . . . .	19
2.2	Motivation . . . . .	19
2.2.1	Built-in behaviors . . . . .	20
2.2.2	Cognitive effort . . . . .	21
2.2.3	Attention . . . . .	22
2.3	Visual processing . . . . .	22
2.3.1	Reading . . . . .	24
2.4	Memory systems . . . . .	25
2.4.1	Short term memory . . . . .	27
2.4.2	Episodic memory . . . . .	29
2.4.3	Recognition and recall . . . . .	29
2.4.4	Serial order information . . . . .	30
2.4.5	Forgetting . . . . .	31
2.5	Learning and experience . . . . .	31
2.5.1	Belief . . . . .	34
2.5.2	Expertise . . . . .	34
2.5.3	Category knowledge . . . . .	36
2.5.4	Categorization consistency . . . . .	38
2.6	Reasoning . . . . .	39
2.6.1	Deductive reasoning . . . . .	40
2.6.2	Linear reasoning . . . . .	41
2.6.3	Causal reasoning . . . . .	42
2.7	Number processing . . . . .	43
2.7.1	Numeric preferences . . . . .	44
2.7.2	Symbolic distance and problem size effect . . . . .	46
2.7.3	Estimating event likelihood . . . . .	46
2.8	High-level functionality . . . . .	47
2.8.1	Personality & intelligence . . . . .	47
2.8.2	Risk taking . . . . .	47
2.8.3	Decision-making . . . . .	48
2.8.4	Expected utility and Prospect theory . . . . .	49
2.8.5	Overconfidence . . . . .	50
2.8.6	Time discounting . . . . .	51
2.8.7	Developer performance . . . . .	51

2.8.8 Miscellaneous . . . . .	53
<b>3 Cognitive capitalism</b>	<b>55</b>
3.1 Introduction . . . . .	55
3.2 Investment decisions . . . . .	56
3.2.1 Discounting for time . . . . .	57
3.2.2 Taking risk into account . . . . .	57
3.2.3 Incremental investments and returns . . . . .	58
3.2.4 Investment under uncertainty . . . . .	59
3.2.5 Real options . . . . .	60
3.3 Capturing cognitive output . . . . .	61
3.3.1 Intellectual property . . . . .	62
3.3.2 Bumbling through life . . . . .	64
3.3.3 Expertise . . . . .	65
3.4 Group dynamics . . . . .	66
3.4.1 Social status . . . . .	67
3.4.2 Social learning . . . . .	67
3.4.3 Group learning and forgetting . . . . .	69
3.4.4 Information asymmetry . . . . .	70
3.4.5 Moral hazard . . . . .	70
3.4.6 Group survival . . . . .	70
3.4.7 Group problem solving . . . . .	72
3.4.8 Cooperative competition . . . . .	73
3.5 Company economics . . . . .	73
3.5.1 Cost accounting . . . . .	74
3.5.2 The shape of money . . . . .	74
3.5.3 Valuing software . . . . .	75
3.6 Maximizing ROI . . . . .	75
3.6.1 Value creation . . . . .	76
3.6.2 Product/service pricing . . . . .	76
3.6.3 Predicting sales volume . . . . .	77
3.6.4 Managing customers as investments . . . . .	79
3.6.5 Hedonism . . . . .	79
3.6.6 Commons-based peer-production . . . . .	80
<b>4 Ecosystems</b>	<b>81</b>
4.1 Introduction . . . . .	81
4.1.1 Funding . . . . .	82
4.1.2 Hardware . . . . .	83
4.2 Evolution . . . . .	85
4.2.1 Diversity . . . . .	86
4.2.2 Lifespan . . . . .	88
4.2.3 Entering a market . . . . .	89
4.3 Population dynamics . . . . .	90
4.3.1 Growth processes . . . . .	91
4.3.2 Estimating population size . . . . .	92
4.4 Organizations . . . . .	94
4.4.1 Customers . . . . .	94
4.4.2 Culture . . . . .	95
4.4.3 Software vendors . . . . .	97
4.4.4 Career paths . . . . .	97
4.5 Applications and Platforms . . . . .	98
4.5.1 Platforms . . . . .	99
4.5.2 Pounding the treadmill . . . . .	100
4.5.3 Users' computers . . . . .	101
4.6 Software development . . . . .	101
4.6.1 Programming languages . . . . .	102
4.6.2 Libraries and packages . . . . .	104
4.6.3 Tools . . . . .	106
4.6.4 Information sources . . . . .	107

<b>5 Projects</b>	<b>109</b>
5.1 Introduction . . . . .	109
5.1.1 Project pecking-order . . . . .	111
5.1.2 Cancellation . . . . .	112
5.2 Resource estimation . . . . .	112
5.2.1 Estimation models . . . . .	114
5.2.2 Money . . . . .	116
5.2.3 Time . . . . .	116
5.2.4 Size . . . . .	117
5.2.5 Software reuse . . . . .	117
5.3 Pitching for projects . . . . .	118
5.3.1 Contracts . . . . .	120
5.4 Paths to delivery . . . . .	121
5.4.1 Development methodologies . . . . .	122
5.4.2 Managing progress . . . . .	124
5.4.3 Major activities . . . . .	125
5.4.4 Discovering functionality needed for acceptance . . . . .	126
5.4.5 Implementation . . . . .	128
5.4.6 Documentation . . . . .	129
5.4.7 Deployment . . . . .	130
5.4.8 Refactoring . . . . .	131
5.5 Development teams . . . . .	131
5.5.1 New staff . . . . .	132
5.6 Product updates . . . . .	133
5.6.1 Project lifespan . . . . .	135
<b>6 Reliability</b>	<b>137</b>
6.1 Introduction . . . . .	137
6.1.1 It's not a fault, it's a feature . . . . .	139
6.1.2 Why do faults occur? . . . . .	140
6.1.3 Reported fault data . . . . .	140
6.1.4 Cultural outlook . . . . .	141
6.2 The search for profit . . . . .	142
6.3 Experiencing a fault . . . . .	145
6.3.1 Input profile . . . . .	145
6.3.2 Further fault experiences—closed population . . . . .	147
6.3.3 Further fault experiences—open population . . . . .	149
6.4 Where is the mistake? . . . . .	151
6.4.1 Requirements . . . . .	151
6.4.2 Source code . . . . .	152
6.4.3 Build system . . . . .	154
6.4.4 Third-party libraries . . . . .	154
6.4.5 Correctness vs. consistency . . . . .	155
6.4.6 Documentation . . . . .	155
6.5 Non-software causes of unreliability . . . . .	155
6.5.1 System availability . . . . .	156
6.6 Checking for intended behavior . . . . .	157
6.6.1 Review meetings . . . . .	158
6.6.2 Testing . . . . .	159
6.6.2.1 Combinatorial testing . . . . .	160
6.6.2.2 Beta testing . . . . .	161
6.6.2.3 Estimating test effectiveness . . . . .	161
6.6.3 Cost of testing . . . . .	162
6.6.4 Runtime issues . . . . .	162
<b>7 Source code</b>	<b>163</b>
7.1 Introduction . . . . .	163
7.1.1 Quantity of source . . . . .	165
7.2 Desirable characteristics . . . . .	166
7.2.1 Programming languages . . . . .	166

7.2.2	Understanding code . . . . .	168
7.2.3	Memory for material read . . . . .	169
7.2.4	Integrating information . . . . .	171
7.2.5	Visual organization . . . . .	173
7.2.6	Identifier names . . . . .	174
7.2.7	Consistency . . . . .	177
7.2.8	Folklore metrics . . . . .	178
7.3	Patterns of usage . . . . .	178
7.3.1	Language features . . . . .	180
7.3.2	Runtime characteristics . . . . .	182
7.3.3	Narrative structures . . . . .	182
7.3.4	Multiple languages . . . . .	183
7.3.5	Control flow . . . . .	183
7.3.6	Statements . . . . .	184
7.3.6.1	Literal values . . . . .	184
7.3.6.2	Calls . . . . .	184
7.3.6.3	Use of variables . . . . .	185
7.3.7	Declarations . . . . .	185
7.3.7.1	Visibility . . . . .	186
7.3.7.2	Ordering of definitions in aggregate types . . . . .	186
7.3.7.3	Unused variables . . . . .	186
7.4	Building programs . . . . .	187
7.5	Evolution of source code . . . . .	187
7.5.1	Database schema . . . . .	188
7.5.2	Source code lifetime . . . . .	188
<b>8</b>	<b>Stories told by data</b>	<b>191</b>
8.1	Introduction . . . . .	191
8.2	Finding patterns in data . . . . .	192
8.2.1	Initial data exploration . . . . .	193
8.2.2	Guiding the eye through data . . . . .	195
8.2.3	Smoothing data . . . . .	196
8.2.4	Densely populated measurement points . . . . .	197
8.2.5	Visualizing a single column of values . . . . .	199
8.2.6	Relationships between items . . . . .	200
8.2.7	3-dimensions . . . . .	200
8.3	Communicating a story . . . . .	202
8.3.1	What kind of story? . . . . .	204
8.3.2	Technicalities should go unnoticed . . . . .	206
8.3.2.1	People have color vision . . . . .	206
8.3.2.2	Color palette selection . . . . .	206
8.3.2.3	Plot axis: what and how . . . . .	207
8.3.3	Communicating numeric values . . . . .	208
8.3.3.1	Percentages vs. frequencies . . . . .	209
<b>9</b>	<b>Probability</b>	<b>211</b>
9.1	Introduction . . . . .	211
9.1.1	Useful rules of thumb . . . . .	212
9.1.2	Measurement scales . . . . .	213
9.2	Probability distributions . . . . .	214
9.2.1	Comparing probability distributions for equality . . . . .	218
9.3	Fitting a probability distribution to a sample . . . . .	220
9.3.1	Zero-truncated and zero-inflated distributions . . . . .	222
9.3.2	Mixtures of distributions . . . . .	223
9.3.3	Heavy/Fat tails . . . . .	225
9.4	Markov chains . . . . .	225
9.4.1	A Markov chain example . . . . .	226
9.5	Social network analysis . . . . .	228
9.6	Combinatorics . . . . .	228
9.6.1	A combinatorial example . . . . .	229

9.6.2 Generating functions . . . . .	230
<b>10 Statistics for software engineering</b>	<b>231</b>
10.1 Introduction . . . . .	231
10.1.1 Statistical inference . . . . .	232
10.2 Samples and populations . . . . .	232
10.2.1 Effect-size . . . . .	233
10.2.2 Sampling error . . . . .	235
10.2.3 Statistical power . . . . .	236
10.3 Describing a sample . . . . .	238
10.3.1 A central location . . . . .	238
10.3.2 Sensitivity of central location algorithms . . . . .	240
10.3.3 Geometric mean . . . . .	240
10.3.4 Harmonic mean . . . . .	241
10.3.5 Contaminated distributions . . . . .	241
10.3.6 Meta-Analysis . . . . .	242
10.4 Statistical error . . . . .	243
10.4.1 Hypothesis testing . . . . .	243
10.4.2 p-value . . . . .	245
10.4.3 Confidence intervals . . . . .	245
10.4.4 The bootstrap . . . . .	246
10.4.5 Permutation tests . . . . .	247
10.5 Comparing samples . . . . .	247
10.5.1 Regression modeling . . . . .	248
10.5.2 Comparing sample means . . . . .	249
10.5.3 Comparing standard deviation . . . . .	253
10.5.4 Correlation . . . . .	254
10.5.5 Contingency tables . . . . .	255
10.5.6 ANOVA . . . . .	256
<b>11 Regression modeling</b>	<b>259</b>
11.1 Introduction . . . . .	259
11.2 Linear regression . . . . .	260
11.2.1 Scattered measurement values . . . . .	263
11.2.2 Discrete measurement values . . . . .	264
11.2.3 Uncertainty only exists in the response variable . . . . .	265
11.2.4 Modeling data that curves . . . . .	268
11.2.5 Visualizing the general trend . . . . .	270
11.2.6 Influential observations and Outliers . . . . .	271
11.2.7 Diagnosing problems in a regression model . . . . .	273
11.2.8 A model's goodness of fit . . . . .	274
11.2.9 Low signal-to-noise ratio . . . . .	275
11.2.10 Abrupt changes in a sequence of values . . . . .	277
11.3 Moving beyond the default Normal error . . . . .	278
11.3.1 Count data . . . . .	279
11.3.2 Continuous response variable having a lower bound . . . . .	281
11.3.3 Transforming the response variable . . . . .	281
11.3.4 Binary response variable . . . . .	282
11.3.5 Multinomial data . . . . .	283
11.3.6 Rates and proportions response variables . . . . .	284
11.4 Multiple explanatory variables . . . . .	285
11.4.1 Interaction between variables . . . . .	288
11.4.2 Correlated explanatory variables . . . . .	290
11.4.3 Penalized regression . . . . .	293
11.5 Non-linear regression . . . . .	294
11.5.1 Power laws . . . . .	297
11.6 Mixed-effects models . . . . .	299
11.7 Generalised Additive Models . . . . .	301
11.8 Miscellaneous . . . . .	303
11.8.1 Advantages of using lm . . . . .	303

11.8.2 Very large datasets . . . . .	303
11.8.3 Alternative residual metrics . . . . .	303
11.8.4 Quantized regression . . . . .	304
11.8.5 Network data . . . . .	304
11.8.6 Solving systems of equations . . . . .	304
11.8.7 Communicating fitted model details . . . . .	304
11.9 Item response theory . . . . .	304
11.9.1 Agreement between raters . . . . .	304
11.10 Time series . . . . .	305
11.10.1 Cleaning time series data . . . . .	306
11.10.2 Modeling time series . . . . .	306
11.10.2.1 Building an ARMA model . . . . .	307
11.10.3 Non-constant variance . . . . .	310
11.10.4 Long-memory processes . . . . .	311
11.10.5 Smoothing and filtering . . . . .	311
11.10.6 Missing data . . . . .	311
11.10.7 Spectral analysis . . . . .	311
11.10.8 Relationships between time series . . . . .	312
11.10.9 Regression models . . . . .	312
11.10.10 Stochastic processes . . . . .	313
11.10.11 Misc . . . . .	313
11.11 Survival analysis . . . . .	313
11.11.1 Kinds of censoring . . . . .	314
11.11.1.1 Input data format . . . . .	315
11.11.2 Survival curve . . . . .	315
11.11.3 Regression modeling . . . . .	316
11.11.3.1 Cox proportional-hazards model . . . . .	317
11.11.3.2 Time varying explanatory variables . . . . .	319
11.11.4 Competing risks . . . . .	322
11.11.5 Multi-state models . . . . .	322
11.12 Circular statistics . . . . .	323
11.12.1 Circular distributions . . . . .	324
11.12.2 Fitting a regression model . . . . .	325
11.12.2.1 Linear response with a circular explanatory variable . . . . .	325
11.12.2.2 Circular response variable . . . . .	326
11.13 Compositions . . . . .	326
<b>12 Miscellaneous techniques</b>	<b>329</b>
12.1 Introduction . . . . .	329
12.2 Machine learning . . . . .	329
12.2.1 Decision trees . . . . .	330
12.3 Clustering . . . . .	332
12.3.1 Sequence mining . . . . .	332
12.4 Ordering of items . . . . .	333
12.4.1 Seriation . . . . .	333
12.4.2 Preferred ordering . . . . .	334
12.5 Simulation . . . . .	335
<b>13 Experiments</b>	<b>337</b>
13.1 Introduction . . . . .	337
13.1.1 Measurement uncertainty . . . . .	338
13.2 Design of experiments . . . . .	338
13.2.1 Subjects . . . . .	340
13.2.2 The task . . . . .	341
13.2.3 What is actually being measured? . . . . .	341
13.2.4 Adapting an ongoing experiment . . . . .	342
13.2.5 Selecting experimental options . . . . .	343
13.2.6 Factorial designs . . . . .	344
13.3 Benchmarking . . . . .	345
13.3.1 Following the herd . . . . .	347

13.3.2 Variability in today's computing systems . . . . .	347
13.3.2.1 Hardware variation . . . . .	348
13.3.2.2 Software variation . . . . .	351
13.3.3 End user systems . . . . .	354
13.3.4 The cloud . . . . .	354
13.4 User interface testing . . . . .	354
13.5 Surveys . . . . .	354
<b>14 Data preparation</b>	<b>357</b>
14.1 Introduction . . . . .	357
14.1.1 Documenting cleaning operations . . . . .	358
14.2 Outliers . . . . .	359
14.3 Malformed file contents . . . . .	360
14.4 Missing data . . . . .	361
14.4.1 Handling missing values . . . . .	362
14.4.2 NA handling by library functions . . . . .	363
14.5 Restructuring data . . . . .	363
14.5.1 Reorganizing rows/columns . . . . .	363
14.6 Miscellaneous issues . . . . .	364
14.6.1 Application specific cleaning . . . . .	364
14.6.2 Different name, same meaning . . . . .	364
14.6.3 Multiple sources of signals . . . . .	365
14.6.4 Duplicate data . . . . .	365
14.6.5 Default values . . . . .	365
14.6.6 Resolution limit of measurements . . . . .	366
14.7 Detecting fabricated data . . . . .	366
<b>15 Overview of R</b>	<b>367</b>
15.1 Your first R program . . . . .	367
15.2 Language overview . . . . .	368
15.2.1 Differences between R and widely used languages . . . . .	368
15.2.2 Objects . . . . .	369
15.3 Operations on vectors . . . . .	370
15.3.1 Creating a vector/array/matrix . . . . .	370
15.3.2 Indexing . . . . .	370
15.3.3 Lists . . . . .	371
15.3.4 Data frames . . . . .	372
15.3.5 Symbolic forms . . . . .	373
15.3.6 Factors and levels . . . . .	373
15.4 Operators . . . . .	374
15.4.1 Testing for equality . . . . .	375
15.4.2 Assignment . . . . .	375
15.5 The R type (mode) system . . . . .	376
15.5.1 Converting the type (mode) of a value . . . . .	377
15.6 Statements . . . . .	377
15.7 Defining a function . . . . .	377
15.8 Commonly used functions . . . . .	378
15.9 Input/Output . . . . .	378
15.9.1 Graphical output . . . . .	379
15.10 Non-statistical uses of R . . . . .	379
15.11 Very large datasets . . . . .	380

# Read me 1st

This book aims to discuss all of what is currently known about software engineering, based on an analysis of all publicly available software engineering data.

This aim is not as ambitious as it sounds because there is not a great deal of data publicly available. Until recently researchers in software engineering concentrated on producing work that gave readers mathematical orgasms, rather than something that might be useful to industry based on experimental evidence.

As work on the book progressed, it became obvious that the best way to organise the material was as two parts, one covering software engineering and the second the statistics used in the analysis of software engineering data.

Life would have been easier if your author could have pointed readers at other books to learn about statistics. Unfortunately existing statistics books are not suitable, for reasons which include:

- they contain too much implementation detail. Developers are casual users of statistics and don't want to spend time learning lots of mathematics; they want to use the techniques, not implement them, and are only interested in the pre and post conditions,
- they target the largest market for introductory statistics books, the social sciences. The characteristics of the data encountered in social sciences are very different from the data encountered in software engineering, e.g., the Normal distribution is commonly encountered in social science data and is not that common in software engineering data, while the exponential distribution is common in software engineering and much less common in the social sciences,
- they continue to use statistical techniques that were designed to be practical for manual implementation (because electronic computers had not yet been invented). If fast computers are available, it is possible to use more powerful techniques which are impractical for manual implementation.

It is assumed that developer time is expensive and computer time is cheap. Where possible a single, general, statistical technique is described and a single way of coding something in R is used. This minimal, but general approach focuses on what developers need to know and the price paid is that the R code may be slower (in many cases the performance slowdown is unlikely to be noticeable).

The approach used is similar to that of a Doctor examining a patient for symptoms that can be matched against known underlying processes.

In some cases the questions I have asked about a particular set of measurements and the techniques used are very different from those made by the researchers who made the original measurements. Reasons for this include needing to illustrate a particular technique and making use of more powerful techniques to extract more information.

Some of the software engineering material, created on the first pass has the feel of a sequence of dots. A second pass is now underway to connect these dots into a coherent whole. The statistics for software engineering material has settled down and hopefully will be useful to people; there are some "...", "???" and places where the document production chain throws a wobbly.

The completed pdf will be made available for free online.

Formatting of large values on graph axis is courtesy of a time-machine from the 1970s; fixing this is on the TODO list.

If you have questions about empirical software engineering that this material does not answer, please let me know.

Even better, if you know of some interesting software engineering data, please tell me where I can download a copy.

All the code and data can be downloaded at: [github.com/Derek-Jones/ESEUR-code-data](https://github.com/Derek-Jones/ESEUR-code-data)

The names of data files usually share the same sequence of initial characters as the pdf file names of the corresponding research paper downloaded from the Internet.



# Chapter 1

## Introduction

Software systems and their host, the electronic computer,<sup>262</sup> began infiltrating the human ecosystem 70+ years ago.<sup>i</sup> Over this period the price of computer equipment continually declined, averaging 17.5% per year,<sup>1562</sup> an economic firestorm that devoured existing practices; software systems are apex predators. Figure 1.1 shows the dramatic fall in the cost of compute operations, however, without cheap mass storage computing would be a niche market; the continuing reduction in the cost of storage created increasing economic incentives for employing the cheap processing power; see Figure 1.2. The scale of improvements in hardware performance (and shortage of trained personnel) were appreciated almost from the beginning.<sup>632</sup>

A shift in perception, from computers as calculating machines,<sup>627</sup> to computing platforms responding in real-time, to multiple independent users, created new problem solving opportunities, e.g., responding in real-time to multiple users; Figure 1.3 shows the initial growth of US based systems capable of sharing cpu time between multiple users.<sup>132</sup>

Since computers were first invented<sup>505,542</sup> people have learned how to do software development through personal experience, it is a craft activity. Most activities that build things start with people learning through their own and sometimes others' experience. In some cases what is being created is important enough that it becomes worth investing in research to discover effective engineering/scientific approaches to the various problems; the benefits of using such an engineering/scientific approach are greater control and predictability.

Why has software engineering not progressed from a craft to an engineering activity?

The engineering/scientific approach is based on theories that have been validated using data obtained from experiments and measurements of events in the world. Until recently measurement data relating to the creation of software systems has either been publicly unavailable or somewhat insubstantial. Over the last decade or so there has been an explosive growth in the collection and analysis of software engineering related data; it is now possible to write a book such as this one.

One reason for the current state of affairs can be found in customer/vendor relationships. Change is driven by those with the power to make it happen, and almost since the beginning, software systems have been a sellers market. The benefits of an engineering approach, are primarily reaped by customers, why should vendors invest in change if customers are willing to continue paying for systems developed using existing practices?

The continuing displacement of existing systems and practices has kept the focus of practice on development (of new systems and major enhancements to existing systems). A change of focus of practice to infrastructure<sup>1574</sup> has to await longer term stability.

Those involved in building systems that use software want to control the process. Control requires understanding; understanding of the many processes involved in building software systems is the goal of software engineering research.

This is a data driven book that treats software engineering as an economic and cultural activity; it is intended to be useful to those involved in building software systems. A topic is only discussed if measurement data is publicly available to ground the discussion. Keeping to

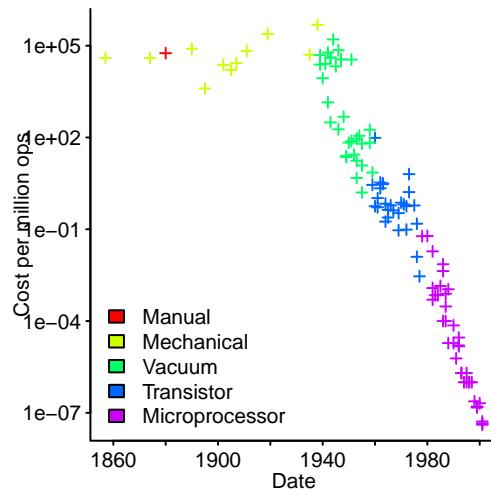


Figure 1.1: Total cost of one million computing operations over time. Data from Nordhaus.<sup>1166</sup> [code](#)

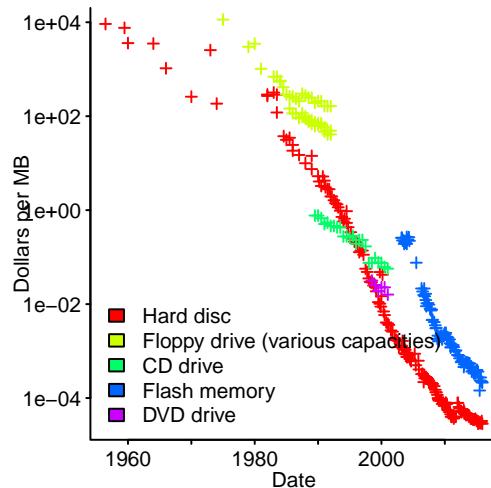


Figure 1.2: Storage cost, in US dollars per Mbyte, of mass market technologies over time. Data from McCallum,<sup>1035</sup> floppy and CD-ROM data kindly provided by Davis.<sup>362</sup> [code](#)

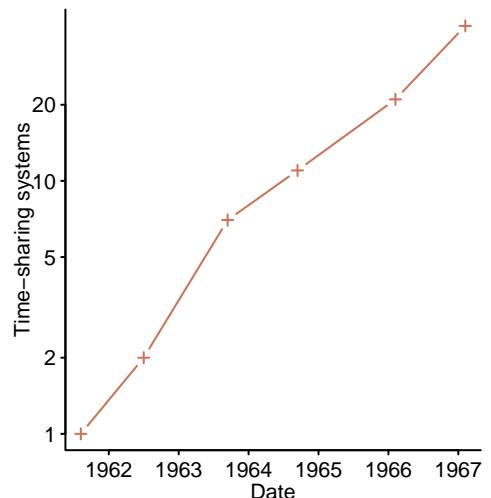


Figure 1.3: Initial growth of time-sharing systems available in the US. Data extracted from Gauthier.<sup>575</sup> [code](#)

<sup>i</sup> The verb "to program" was first used in its modern sense during 1946.<sup>626</sup>

this requirement means that readers are likely to be dismayed at the scant coverage of many topics of importance in software engineering.

Software is written within a particular development culture, by people having their own unique and changeable behavior patterns. Measurements of the products and processes in this environment are intrinsically noisy and are likely to include variables of influence that are not measured. This situation does not mean that analysis of the available measurements is a futile activity, but it does mean that the uncertainty and variability is likely to be much larger than typically found in other engineering disciplines.

Measurement data is analysed using statistics, with statistical techniques being wielded as weaponised pattern recognition; those seeking discussions written in the style of a stimulant for mathematical orgasms, will not find satisfaction here.

Statistics does not assign a meaning to any of the patterns it uncovers; interpreting the patterns thrown up by statistical analysis, to give them meaning, is your job dear reader (based on your knowledge and experience of the problem domain).

The tool used for statistical analysis is the R system.<sup>1299</sup> R was chosen because of its ecosystem; there are many books, covering a wide range of subject areas, using R and active online forums discussing R usage (answers to problems can often be found by searching the Internet or if none are found a question can be posted with a reasonable likelihood of receiving an answer).

The data and R code used in this book are freely available for download from the book's website.<sup>792</sup>

Like programming, data analysis contains a craft component, and the way to improve craft skills is to practice.

In January 2018 the U.S. Congress passed the Foundations for Evidence-Based Policymaking Act of 2018,<sup>322</sup> whose requirements include: "(1) A list of policy-relevant questions for which the agency intends to develop evidence to support policymaking. (2) A list of data the agency intends to collect, use, or acquire to facilitate the use of evidence in policymaking."

## 1.1 Software ecosystems

The last 50 years or so has been a sellers market; the perceived benefits provided by software systems has been so significant that companies that did not use them risked being eclipsed by competitors. Whole industries were engulfed and companies became Red Queens, who lost their position if they failed to keep running.

Provided software development projects looked like they would deliver something that was good enough, everybody knew that the customer would wait and pay; complaints could be ignored. Software vendors learned that the only way to survive in a rapidly evolving market was to get products to market quickly, before things moved on.

Experience from the introduction of earlier high-tech industries suggests that it takes many decades for major new technologies to settle down and reach market saturation.<sup>1234</sup> For instance, the transition from wood to steel for building battleships,<sup>1204</sup> started in 1858 and reached its zenith during the second world war; there is a long history of growth and decline various forms of infrastructure (see Figure 1.4).

Over the last 70 years a succession of companies have dominated the computing industry. The continual reduction in the cost of computing platforms created new markets and occasionally one of these grew to become the largest market, financially, for computing resources. A company dominates the computer industry when it dominates the market that dominates the industry. Companies that lose computing industry dominance often continue to grow, only declining when the market they have dominated declines.

Figure 1.5 illustrates the impact of the growth of new markets on the market capitalization of three companies; IBM dominated when mainframes dominated the computer industry, the desktop market grew to dominate the computer industry and Microsoft dominated, smart phones removed the need for computers sitting on desks, and these have grown to dominate the computer market with Apple being the largest company in this market (Google's Android investment is a defensive move to ensure they are not locked out of advertising on mobile, i.e., it is not as intended to be a direct source of revenue, making it extremely difficult to

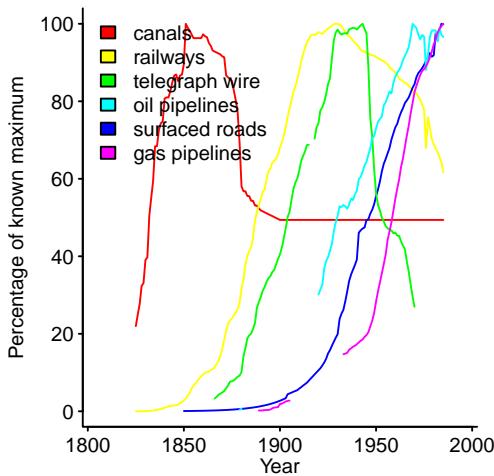


Figure 1.4: Growth of transport and product distribution infrastructure in the USA (underlying data is measured in miles). Data from Grubler et al.<sup>634</sup> code

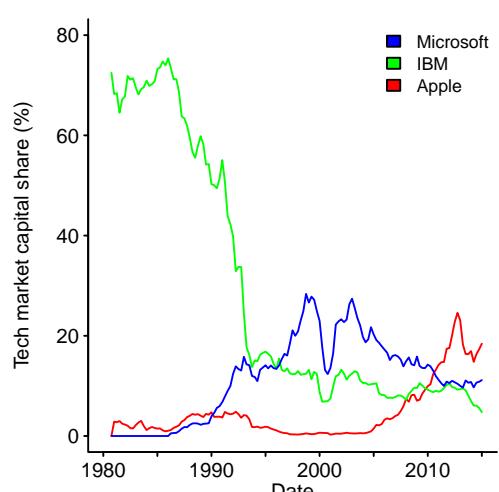
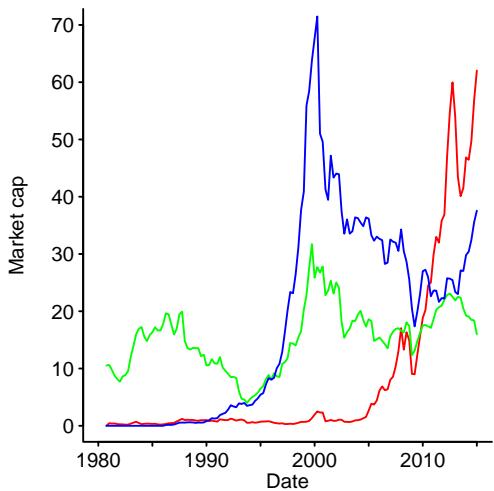


Figure 1.5: Market capitalization of IBM, Microsoft and Apple (upper), and expressed as a percentage of the top 100 listed US tech companies.<sup>429</sup> code

estimate its contribution to Google's market capitalization). It shows market capitalization (upper), and as a percentage of the top 100 listed US tech companies (lower), as of the first quarter of 2015.<sup>429</sup>

The three major eras, each with its own particular product and customer characteristics, have been (Figure 1.6 shows sales of major computing platforms):

- the IBM era (sometimes known as the *mainframe* era, after the dominant computer hardware, rather than the dominant vendor): The focus was on business customers, with high priced computers sold, or rented, to large organizations who either rented software from the hardware vendor or paid for software to be developed specifically for their own needs. Large companies were already spending huge sums employing the (tens of) thousands of clerks needed to process the paperwork used in the control their businesses;<sup>141</sup> large companies could make large savings, in time and money, by investing in bespoke software systems, paying for any subsequent maintenance, and the cost of any faults experienced.

When the actual cost of software faults experienced by one organization is very high (with potential for even greater costs if things go wrong), and the same organization is paying all, or most of the cost of creating the software, that organization can see a problem that is costing it money, that it thinks it should have control over. Very large organizations are in a position to influence research agendas to target the problems they want solved.

Large organizations tend to move slowly. The rate of change was slow enough for experience and knowledge of software engineering to be considered essential to do the job (this is not to say that anybody had to show that their skill was above some minimum level before they would be employed as a software developer).

- the Wintel era: the Personal Computer running Microsoft Windows, using Intel's x86 processor family, was the dominant computing platform. The dramatic reduction in the cost of owning a computer significantly increases the size of the market, and it becomes commercially profitable for companies to create and sell software packages. The direct cost of software maintenance is not visible to these customers, but they pay the costs of the consequences of faults they experience when using the software.

Microsoft's mantra of a PC on every desk required that people write software to cover niche markets. The idea that anyone could create an application was promoted as a means of spreading Windows, by encouraging people with application domain knowledge to create software running under MS-DOS and later Windows.

Programming languages, libraries and user interface experienced high rates of change, which meant that developers found themselves on a learning treadmill. One lesson learned by some was that the likelihood of change did not make it worthwhile investing in becoming an expert in a particular area.

- the Internet era, no single vendor dominates the Internet, but some large niches have dominant vendors, e.g., mobile phones,

It is difficult to judge whether the rate of change has been faster than in previous eras, or the volume of discussion about the changes has been higher because the changes have been visible to more people, or the lack of a dominant vendor to prevent change occurring too quickly.

The ongoing history of new software systems and computing platforms has created an environment where people are willing to invest their time and energy creating what they believe will be the next big thing. Those with the time and energy to do this, are often the young and inexperienced, outsiders in the sense that they don't have any implementation experience with existing systems. If any of these new systems take off, the developers involved will have made, or will make, many of the same mistakes made by the developers involved in earlier systems. The rate of decline of major software platforms is slow enough that employees with significant accumulated experience and expertise can continue to enjoy their seniority in well-paid jobs and have no incentive to jump ship to apply their expertise to an emerging system.

Mobile computing is only commercially feasible when the cost of computation, measured in Watts of electrical power, can be supplied by user-friendly portable batteries. Figure 1.7

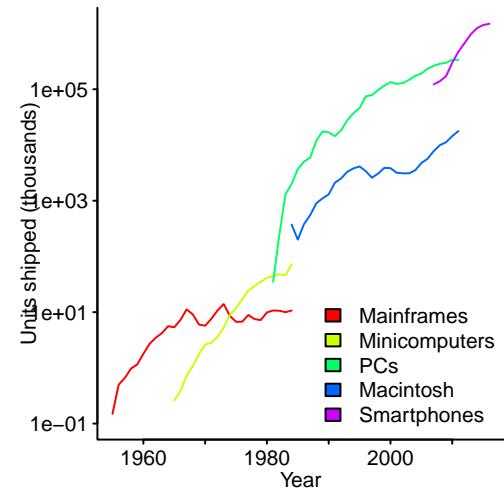


Figure 1.6: Total annual sales of computer species over the last 60 years. Data from Gordon<sup>601</sup> (mainframes and minicomputers), Reimer<sup>1322</sup> (PCs) and Gartner<sup>548</sup> (smartphones). [code](#)

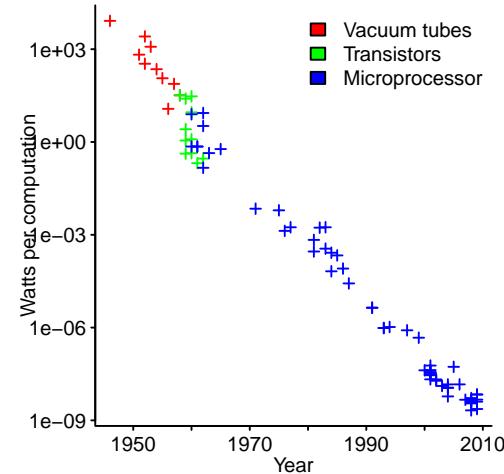


Figure 1.7: Power consumed, in Watts, executing an instruction on a computer available in a given year. Data from Koomey et al.<sup>876</sup> [code](#)

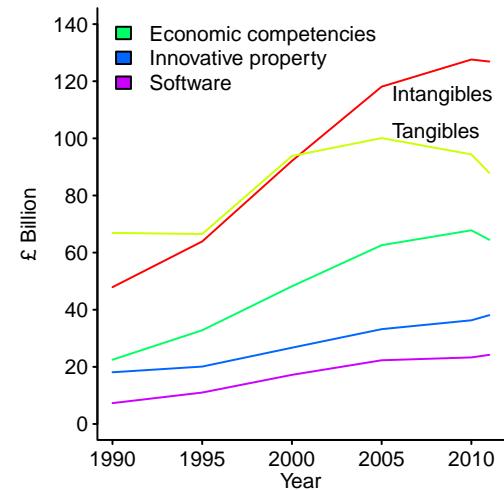


Figure 1.8: Total investment in tangible and intangible assets by UK companies, based on their audited accounts. Data from Goodridge et al.<sup>594</sup> [code](#)

shows the decline in electrical power consumed by a computation between 1946 and 2009; historically, it has been halving every 1.6 years.

Software systems have yet to reach a stable market equilibrium in many of the ecosystems they have colonised. Many software systems are still new enough that they are expected to adapt when the ecosystem in which they operate evolves. The operational procedures of these systems have not yet been sufficiently absorbed into the fabric of life that they enjoy the influence derived from users understanding that the easiest and cheapest option is to change the world to operates around them.

Economic activity is shifting towards being based around intangible goods;<sup>658</sup> cognitive capitalism is becoming mainstream.

A study by Goodridge, Haskel and Wallis<sup>594</sup> estimated the UK investment in intangible assets, as listed in the audited accounts that UK companies are required to file every year. Figure 1.8 shows the total tangible (e.g., buildings, machinery and computer hardware) and intangible assets between 1990 and 2012. Economic competencies are items such as training and branding, Innovative property includes scientific R&D, design and artistic originals (e.g., films, music and book); accounting issues associated with software development are discussed in Chapter 3.

Some companies treat some software as fixed-assets. Figure 1.9 shows quarterly totals for newly purchased and own software, and computer hardware, reported by UK companies as new fixed-assets.

A study by Wang<sup>1630</sup> found that while firms associated with the current IT fashion have a higher reputation and pay their executives more, they do not have higher performance. As companies selling hardware have discovered, designing products to become technologically obsolete (perceived or otherwise)<sup>1455</sup> or wear out and break after a few years,<sup>7</sup> creates a steady stream of sales. Given that software does not wear out, the desire to not to be seen as out of fashion provides a means for incentivizing users to update to the latest version.

Based on volume of semiconductor sales (see Figure 1.10), large new computer-based ecosystems are being created in Asia; the rest of the world appears to have completed their growth phase.

The economics of chip fabrication,<sup>894</sup> from which Moore's law derived,<sup>ii</sup> was what made the firestorm possible; the ability to create more products (by shrinking the size of the components; see Figure 1.11) for roughly the same production cost (of going through the chip fabrication process);<sup>716</sup> faster processors and increased storage capacity on similar-sized devices were fortuitous side effects. The upward ramp of the logistic equation has now levelled off<sup>506</sup> and by the time this book appears Moore's law will be something that people used to write about.

Software systems are part of a world-wide economic system that has been rapidly evolving for several hundred years; the factors driving economic growth are slowly starting to be understood.<sup>781</sup> Analysis of changes in World GDP have found cycles, or waves, of economic activity; Kondratieff waves are the longest, with a period of around 50 years (data from more centuries may contain a longer cycle), a variety of shorter cycles are also seen, such as the Kuznets swing of around 20 years. Five Kondratieff waves have been identified with major world economic cycles, e.g., from the industrial revolution to information technology.<sup>1234</sup> Figure 1.12 shows a spectral analysis of estimated World GDP between 1870 and 2008; adjusting for the two World-wars produces a smoother result.

The rate at which new technologies have spread to different countries has been increasing over time.<sup>316</sup> There has been some lag in the diffusion of software systems<sup>331</sup> across the world.

Computers were the enablers for the latest wave, from the electronic century.

### 1.1.1 The primary activities of software engineering

Software engineering is the collection of activities performed by people involved in producing and maintaining software, executed by computer(s) to solve a problem.

<sup>ii</sup> Moore's original paper,<sup>1105</sup> published in 1965, extrapolated four data-points to 1975 and questioned whether it would be technically possible to continue the trend

These activities will have some path dependency, that is, once the know-how and infrastructure for performing some activity becomes established the existing practice is likely to continue.

Perhaps the most entrenched path dependency in software development is the use of two-valued logic, i.e., binary. The most efficient radix, in terms of representation space (number of digits times number of possible values of each digit), is,  $2.718\dots$ ,<sup>670</sup> the closest integral value is 3. The use of binary, rather than ternary, was driven by the characteristics of the available electronic switching devices.<sup>iii</sup> Given the vast quantity of software making an implicit, and in some cases explicit, assumption that binary representation is used, a future switching technology that would support the use of a ternary representation might not be adopted or be limited to resource constrained environments.<sup>1638</sup>

Traditionally activities have included: obtaining requirements, creating specifications, design at all levels, writing and maintaining code, writing manuals fixing problems and providing user support. Employees within large organizations specialise within a particular area, and these activities are the areas where software developers specialise.

In small companies there is greater opportunity, and sometimes a need, for employees to become involved in tasks that would not be considered part of the job of a software developer in larger companies. For instance, being involved in any or all of a company's activities from the initial sales inquiry through to customer support of the delivered system; the financial aspect of running a business is likely to be much more visible in a small company.

Software is created and used within a variety of ecosystems and software engineering activities can only be understood in the context of the ecosystem in which it operates.

While this is an overly broad definition, let's be ambitious and run with it, allowing the constraints of data availability and completing a book to provide the filtering.

<sup>130</sup> ...

The debate over the identity of computing as an academic discipline is ongoing.<sup>1540</sup>

## 1.2 Brief history of software engineering research

The fact that software often contained many faults and took much longer than expected to produce was a surprise to those working at the start of electronic computing, after World War II. Established practices for measuring and documenting the performance of electronics were in place and ready to be used for computer hardware.<sup>869,1245</sup> It was not until the end of the 1960s that a summary of the major issues appeared in print.<sup>1137</sup>

Until the early 1980s most software systems were developed for large organizations, with over 50% of US government research funding for mathematics and computer science coming from the Department of Defense,<sup>504</sup> an organization that built large systems, over time-frames of many years. As customers of software systems, these organizations have promoted a customer orientated research agenda, e.g., focusing on minimizing customer costs and risks, with no interest in vendor profitability and risk factors. Also, the customer is viewed being a large organization.

Large organizations, such as the DOD, spend so much on software that it is worth their while investing in research, aimed at reducing costs and learning how to better control the development process. During the 1970s project data, funding and management by the Rome Air Development Center,<sup>iv</sup> RADC, came together to produce the first collection of wide-ranging evidence-based reports analysing the factors involved in the development of large software systems.<sup>382,1547</sup>

For whatever reason the data available at RADC was not widely distributed or generally known about; the only people making use of this data in the 1980s and 1990s appear to be Air Force officers writing Master's thesis.<sup>1200,1434</sup>

<sup>iii</sup> In a transistor switch, Off is represented by very low-voltage/high-current and On represented by saturated high-voltage/very low-current. Transistors in these two states consume very little power (power equals voltage times current). A third state would have to be represented at a voltage/current point that would consume significantly more power. Power consumption, or rather the heat generated by it, is a significant limiting factor in processors built using transistors.

<sup>iv</sup> The main US Air Force research lab. There is probably more software engineering data to be found in US Air Force officers' Master's thesis, than all academic software engineering papers published before the early 2000s.

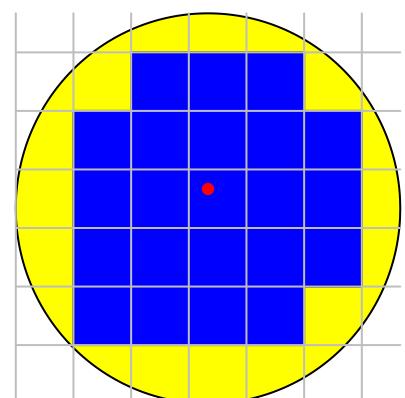
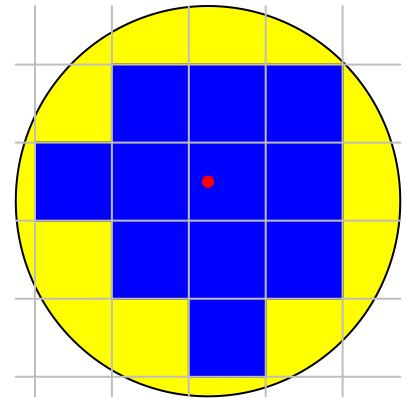


Figure 1.11: Smaller size allows more devices to be fabricated on the same slice of silicon, plus material defects impact a lower percentage of devices. [code](#)

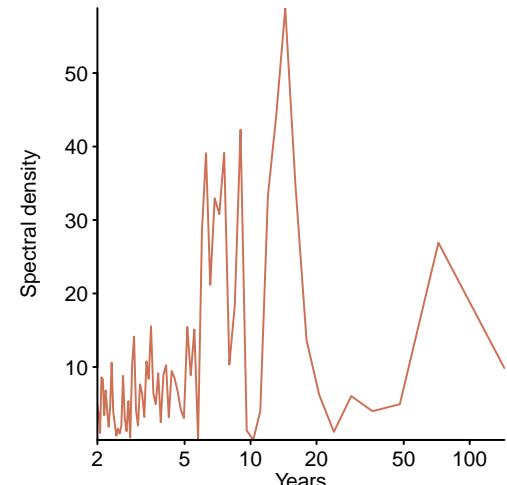


Figure 1.12: Spectral analysis of World GDP between 1870-2008; peaks around 17 and 70 years. Data from Maddison.<sup>1005</sup> [code](#)

The legacy of this first 30 years was a research agenda oriented towards building large software systems.

Most published software engineering research, since around 1980, has not made use of empirical evidence (during this time, not including a theoretical contribution in empirical research was considered grounds for rejecting a paper submitted for publication<sup>11</sup>). A review<sup>986</sup> in the early 1990s, of published papers relating to software engineering, found an almost total lack of evidence-based analysis of its engineering characteristics; a systematic review of 5,453 papers published between 1993 and 2002<sup>655</sup> found 2% reporting experiments. When experiments had been performed, they suffered from small sample sizes<sup>817</sup> (a review<sup>1625</sup> using papers from 2005 found that little had changed), had statistical power falling well below norms used in other disciplines<sup>424</sup> or simply failed to report an effect size (for the 92 controlled experiments published between 1993 and 2002 only 29% reported an effect size<sup>817</sup>).

Why have academics working in an engineering discipline not based their research on empirical data? The difficulty of obtaining realistic data is sometimes cited<sup>1281</sup> as the reason. However, researchers in business schools have been able to obtain production data from companies,<sup>v</sup> perhaps the real reason is those in computing departments are more interested in algorithms and mathematics, rather than the more human factors oriented needed for commercial software development. The publication and research culture in computing departments may have discouraged anybody wanting to do evidence-based work (a few intrepid souls did run experiments using professional developers<sup>109</sup>). Researchers with a talent for software engineering either moving on to other research areas or to working in industry, leaving the field to those with talents in the less employable areas of mathematical theory, literary criticism (of source code) or folklore.<sup>219</sup>

A lack of evidence has not prevented researchers expounding plausible sounding theories that, in some cases, have become widely regarded as true. For instance, it was once claimed, without any evidence, that the use of source code clones (i.e., copying code from another part of the project) is bad practice (e.g., clones are likely to be a source of faults, perhaps because only one of the copies was updated).<sup>522</sup> In practice, research has shown that the opposite is true, <sup>1307, 1554</sup> clones are less likely to contain faults than *uncloned* source.

Given the sparsity of available evidence relating to industrial software systems, it is unsurprising that academics researchers have failed to create any reliable predictive or descriptive software engineering theories having a connection to reality.

Many of the beliefs relating to software engineering processes, commonly encountered in academic publications, are based on ideas created many years ago by researchers who were able to gain access to a relevant (often tiny) dataset. For instance, Perry<sup>1235</sup> divided software interface faults into 15 categories using a data set of just 85 modification requests to draw conclusions; this work is still being cited in papers 25 years later. These fossil theories have continued to exist because of the sparsity of data needed to refute or improve on them.

Human psychology and sociology continue to be completely ignored as major topics of software research, a fact pointed out over 35 years ago.<sup>1420</sup>

Quantity of published papers should not be confused with progress towards an effective model of behavior. For instance there has been a great deal published on software process improvement, SPI, (635 papers were found by a recent study<sup>892</sup> of research over the last 25 years), with experience reports and proposed solutions making up two-thirds of the publications; however, the proposed solutions were barely evaluated, there were no studies evaluating advantages and disadvantages of proposals, and the few testable theories are waiting to be tested.

Over the last 10 years or so there has been an explosion of empirical research, driven by the availability of large amounts of data extracted from open source software. This significant change in the availability of data, and the resulting boom in the fortunes of empirical research does not mean that existing theories and ideas will change overnight. Simply ignoring all research published before 2005 (roughly when the data deluge started) does not help, earlier research has seeded old wives tales that have become embedded in the folklore of software engineering creating a legacy that is likely to be with us for sometime to come.

It is inevitable that some early papers will make claims about software engineering that we now believe to be true. It is possible to search though the contents of old papers looking

---

<sup>v</sup> Many commercial companies do not systematically measure themselves and maintain records, so finding companies that have data requires contacts and persistence.

for wording that can be interpreted as a connection with a recent discovery in the same way that it is possible to search through the prophecies of Nostradamus to find one that can be interpreted as predicting the same discovery.

This book takes the approach that evidence-based theories in software engineering don't yet exist, the subject is a blank slate. Knowing that certain patterns of behavior regularly occur is an empirical observation, a theory would make verifiable predictions that include the observed patterns. In some cases existing old wives tales are discussed if it is felt that their use in an engineering environment would be seriously counter-productive. For instance, while various software metrics (e.g., Halstead's metric) are widely known, your authors' experience is that practicing developers do not invest effort in using them; they are famous for being famous and so there is little to be gained in spending much effort debunking them.

## 1.2.1 Folklore

The dearth of experimental evidence has left a vacuum that has been filled by folklore and old-wives' tales. Examples of software folklore include claims of a 28-to-1 productivity difference between best/worst developers, and that parameter passing in function calls is resource intensive for embedded systems.

The productivity claim, sometimes known as the *Grant and Sackman study*, is based on an incorrect reading of a 1966 study<sup>614</sup> by those authors. This study set out to measure the extent to which the then new time-sharing approach to computer usage, was more productive for software development than existing batch processing systems (where jobs were typically submitted via punched cards, with program executing several hours later, with their output printed on paper).

The paper<sup>614</sup> was published in a widely read journal, and summarised the data; a table listed the ratio between the best subject performance using the fastest system, and the worst subject performance on the slowest system; the 28:1 ratio included programmer and systems performance differences (if batch/time-sharing differences are separated out the maximum difference ratio is 14:1, the minimum 6:1). The actual measurement data was published in a relatively inaccessible journal,<sup>1367</sup> see Figure 8.22.

In 1981, a study by Dickey<sup>401</sup> separated out subject performance from other factors, adjusted for individual subject differences, and found a performance difference ratio of 1:5. However, by 1982 the 28:1 ratio had appeared in widely read magazine articles and books, and had become established as *fact*. In 1999, a study by Prechelt<sup>1277</sup> explained what had happened, for a new audience.

Embedded software runs on resource limited hardware, which is often mass-produced, and saving pennies per device can add up to a lot of money; systems are populated with the smallest possible memory and power consumption is reduced by using the slowest possible clock speeds, e.g., closer to 1 MHz than 1 GHz.

Experienced embedded developers are aware of the hardware performance limitations they have to work within. Many low cost processor have a very simple architecture with relatively few instructions and parameter passing, to a function, can be very expensive (in execution time and code size) compared to passing values to functions in global variables on some processors.

A study by Engblom<sup>446</sup> investigated differences in the characteristics of embedded C software, and the SPECint95 benchmark. Figure 1.14 shows the percentage of function definitions containing a given number of parameters, for embedded software the SPECint95 benchmark and desktop software measured by Jones.<sup>785</sup> A Poisson distribution provides a reasonable fit to both sets of data; for desktop software, the distribution of function definitions having a given number of parameters the Poisson distribution has  $\lambda = 2$ , while for embedded developers  $\lambda = 0.8$ .

These measurements were of source code from the late 1990s, are things likely to have changed?

Companies are likely to be just as interested in saving pennies. Compilers may have become better at reducing function parameter overheads for some processor, however, it is believed that drives developer usage.

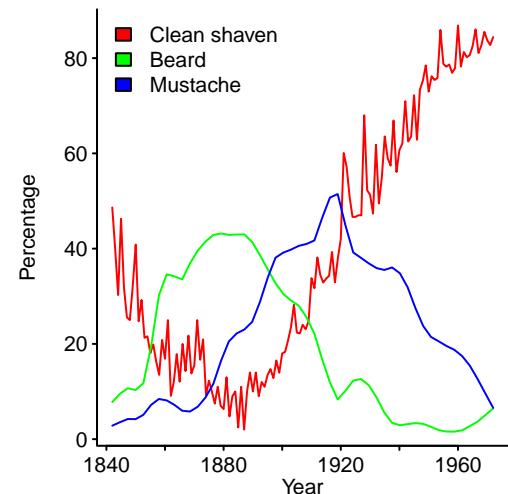


Figure 1.13: Changing habits in men's facial hair. Data from Robinson.<sup>1339</sup> [code](#)

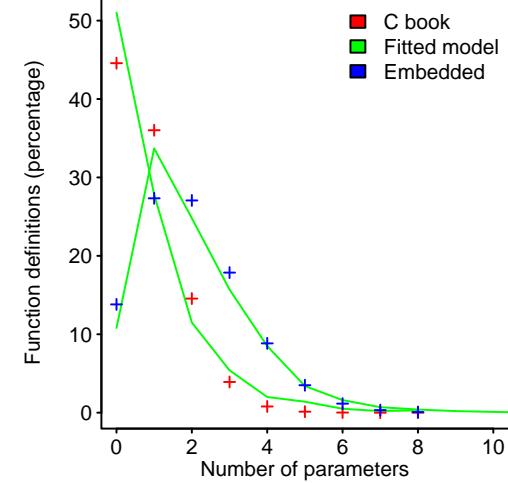


Figure 1.14: Percentage of function definitions declared to have a given number of parameters in: embedded applications, the SPECint95 benchmark???, and the translated form of C source benchmark programs. Data for embedded and SPECint95 kindly supplied by Engblom,<sup>446</sup> C book data from Jones.<sup>785</sup> [code](#)

Embedded devices have become more mainstream, with companies selling IoT devices with USB interfaces. This availability provides an opportunity for aspects of the culture of developing for desktop and mobile systems to invade the culture of embedded development. In situations where code size or/and performance is critical, developers looking for savings may learn about the overheads of parameter passing.

Is the values of  $\lambda$  always approximately 0.8 or 2.0? Perhaps there is a range of values, depending on developer experience (old habits die hard and parameter overhead will depend on processor characteristics, e.g., 4-bit, 8-bit and 16-bit processors).

## 1.2.2 The academic ecosystem

Interactions between people in industry and academia has often suffered from a misunderstanding of each other's motivations and career pressures.

Few people in industry have much interaction with the academic ecosystem that claims to research their field. The quaint image of researchers toiling away for years before publishing a carefully crafted manuscript is long gone.<sup>433</sup> Although academics continue to work in a feudal based system of patronage and reputation, they are incentivised by the motto “publish or perish”,<sup>1220</sup> with science perhaps advancing one funeral at a time.<sup>75</sup> Hopefully the change to evidence-based software engineering research will progress faster than fashions in men’s facial hair (most academics researching software engineering are men); see Figure 1.13.

Academic research projects share many of the characteristics of commercial start-ups. They involve a few people attempting to solve a sometimes fuzzily defined problem, trying to make an improvement in one area of an existing *product*, and they often fail, with the few that succeed producing spectacular returns. Researchers are serial entrepreneurs in that they tend to only work on funded projects, moving onto other projects when funding runs out (and often having little interest in previous projects). Like commercial product development, the choice of research topics is fashion driven; see Figure 1.15.

The visible output from academic research are papers published in journals and conference proceedings. It is important to remember that: “ . . . an article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment, and the complete set of instructions which generated the figures.”<sup>224</sup>

Many journals and conferences uses a process known as *peer review* to decide whether a submitted paper should be accepted. The peer review process involves sending submitted papers to a few referees, ideally chosen for their expertise of the topic covered, who make suggestions on how to improve the paper and provide a yes/no acceptance decision (the identity of the paper’s author(s), and the referees are often anonymous). The peer review process has generally worked well in the past, but is looking to be in need of a major overhaul<sup>1279</sup> to improve turn-around time and handle the work load generated by a proliferation of journals and conferences; some journals and conferences have become known for the low quality of their peer reviews. The status attached to the peer review process has resulted in it being adopted by journals covering non-traditional fields, e.g., psychic research.

In the past most researchers have made little, if any, effort to archive the data they gather for future use. One study<sup>1614</sup> requested the datasets relating to 516 biology related studies, with ages between 2 and 22 years; they found that the probability of the dataset being available fell by 17% per year. Your author’s experience of requesting data from researchers is that it often fails to survive beyond the lifetime of the computer on which it was originally held.

Researchers may have reasons, other than carelessness, for not making their data generally available. For instance, a study<sup>1659</sup> of 49 papers in major psychology journals found that the weaker the evidence for the researcher’s hypothesis the less likely they were to be willing to share their data.

The importance of making code and data available is becoming widely acknowledged, with a growing number of individual researchers making code/data available for download and journals having explicit policies about authors making this information available.<sup>1510</sup> In the UK researchers who receive any funding from a government research body are now required to archive their data and make it generally available.<sup>1326</sup>

Discovering an interesting pattern in experimental data is the start, not the end of experimentation involving that pattern (the likelihood of obtaining a positive result is as much to do

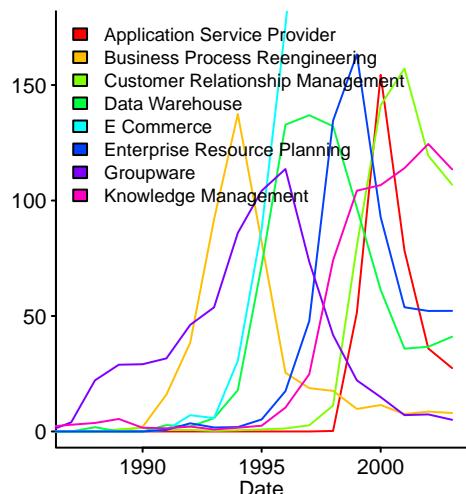


Figure 1.15: Number of papers, in each year between 1987 and 2003, associated with a particular IT topic. The E-commerce paper count peaks at 1,775 in 2000 and in 2003 is still off the scale compared to other topics. Data kindly provided by Wang.<sup>1630</sup> [code](#)

with the subject studied as the question being asked<sup>468</sup>). The more often behavior is experimentally observed the greater the belief that the effect seem actually exists. Replication is the process of duplicating the experiment(s) described in a report or paper to confirm the findings previously obtained. For replication to be practical, researchers need to be willing to share their code and data, something that a growing numbers of software engineering researchers are starting to do; those attempting replication are meeting with mixed success.<sup>315</sup>

Replication is not a high status activity, with high impact journals interested in publishing research which reports new experimental findings and not wanting to dilute their high impact status by publishing replications (which, when they are performed and fail to replicate previous results considered to be important discoveries, can often only find a home for publication in a low impact journal). A study<sup>1190</sup> replicating 100 psychology experiments found that while 97% of the original papers reported p-values less than 0.05, only 36% of the replicated experiments obtained p-values this low; a replication study<sup>268</sup> of 67 economics papers was able to replicate 22 (33%) without assistance from the authors, 29 with assistance.

Replication is necessary, i.e., the results claimed by one researcher must be reproduced by another if they are to be accepted. Without replication researchers are amassing a graveyard of undead theories.<sup>489</sup>

These days academic performance is often measured by number of papers published, and the impact factor of the journal in which they were published<sup>897</sup> (scientific journals publish a percentage of the papers submitted to them, with prestigious high impact journals publishing a lower percentage than those with lower impact factors; journals and clubs share a common economic model<sup>1273</sup>). Organizations that award grants to researchers often consider the number of published papers and impact factor of the publication journal when deciding whether to award a grant; the effect is to generate an evolutionary pressure that selects for bad science.<sup>1463</sup>

One consequence of the important role of published paper count in an academic's career is an increase in scientific fraud,<sup>467</sup> most of which goes undetected;<sup>313</sup> one study<sup>469</sup> found that most retracted papers (67.4%) were attributable to misconduct, around a 10-fold increase since 1975. More highly ranked journals have also been found to contain a higher percentage of retracted papers,<sup>206</sup> it is not known whether this represents an increased in flawed articles, or an increase in detection.<sup>1497</sup> Only a handful of software engineering papers have been retracted, perhaps because a lack of data makes it very difficult to verify any of the claims made by the authors. The website [retractionwatch.com](http://retractionwatch.com) reports on possible and actual paper retractions.

**Commercial research labs** Many large companies have research groups. While the researchers working in these groups often attend the same conferences as academics and publish papers in the same journals; their performance is often measured by the number of patents granted.

**Citation practices** This book attempts to provide citations to any claims made, so that readers can perform background checks. To be cited papers have to be freely available for public download, unless published before 2000 (or so), and when data is analysed it has to be free for public distribution.<sup>vi</sup>

When academics claim that their papers can be freely downloaded what they may mean is that the university that employs them has paid for a site-wide subscription that enables university employees to download copies of papers published by various commercial publishers. Tax payers pay for the research and university subscriptions and may not have free access to it. Things are slowly changing. In the UK researchers in receipt of government funding are now incentivized to publish in journals that will make papers produced by this research freely available after a period of 6-12 months from publication date. Your author's attitude is that academics are funded by tax payers and if they are unwilling to provide a freely downloadable copy on their web page, they should not receive any credit for the work.<sup>vii</sup>

Software developers are accustomed to treating documents that are more than a few years old, as being out-of-date; a consequence of the fast changing environment in which they operate. Some fields, such as cognitive psychology, are more established and people do not feel the need to keep repeating work that was first performed decades ago (although it may

<sup>vi</sup> The usual academic procedure is to cite the first paper that proposes a new theory, describes a pattern of behavior, etc.

<sup>vii</sup> In fact they should not have been funded in the first place; if an academic refuses to make a copy of their papers freely available to you, please report their behavior to your elected representative.

be replicated as part of the process of testing new hypothesis). In more established fields, it is counter-productive to treat date of publication as a worthiness indicator.

?

## 1.3 Lessons learned from the analysis in this book

The gold standard of the scientific method is the controlled randomised experiment, followed by replication of the results by others (failure to replicate the finding of earlier studies is common<sup>753</sup>). In this kind of experiment, all the factors that could influence the outcome of an experiment are either controlled or randomly divided up such that any unknown effects add noise to the signal rather than ghost patterns.

A handful of the ?510? datasets analysed in this book came from controlled experiments, some of which used randomization, and a handful of this handful have been replicated.

Open source is readily available in quantity, and a lot of empirical research now makes use of this public resource. To what extent are results obtained from measurements on Open Source projects applicable to non-Open Source software development, i.e., how closely do the characteristics of open source match the characteristics of closed source? In some cases there may be little reason to expect any differences, while in others large differences may exist (e.g., the speed with which developers respond to security advisories<sup>893</sup> is likely to be faster in a commercial environment). The only definitive answer comes from comparing data from both environments.

Other possible differences might arise from Open Source software developers having a larger say in what features are implemented, rather the manager or individual clients; successful Open Source projects probably have many more users than most commercial systems (which may only be used internally or have a few large customers)...

The patterns of behavior seen in the analysis discussed in this book should be treated as suggestions for what might be. Perhaps their most practical application is in helping to dismiss behaviors that might otherwise be thought possible.

and...

The measurement process invariably introduces some amount of error. While it may be possible to measure source code to a high degree of accuracy, other the values of software related quantities are likely to be a lot less certain. Table 1.1 lists estimates of the value of U.S. Computer manufacturer shipments, made by different data gathering organizations.

Authority	Worldwide			Domestic		
	1960	1965	1970	1960	1965	1970
EIA	0.630	2.830	5.162		2.575	3.958
EDP/IR	0.72	2.40	7.29	0.59	1.77	4.37
ADL				7.27	0.53	2.10
						4.94

Table 1.1: Estimated value of shipments by U.S. computer manufacturers, in \$billion, made by various data gathering organizations. Data from Phister<sup>1245</sup> table 0.2.

## 1.4 Overview of contents

This book has two parts:

- a discussion of the major components of software engineering, driven by an analysis of the publicly available data. Many topics usually covered in software engineering textbooks are not discussed because public data relating to them could not be located,
- a discussion of the data analysis techniques applicable to the kinds of measurement data and problems likely to be encountered by professional software developers. This material provides the background to the data analysis in the first part.

The intent is to provide pointers to data analysis and experimental techniques that might be used to obtain and analyse measurement data; also, to provide guidance for solving a data driven problem in software engineering.

While readers will learn something about data analysis, the material is not intended to provide an introduction to data analysis as such, but rather an introduction to the use of data analysis concepts and methods, along with the assumptions that underlie them. A suggested reading list is provided...

**Human cognitive characteristics** The operating characteristics of the human brain is an essential component of any study of software engineering. Characteristics such as cognitive capacity, memory storage/recall performance and learning, personality (e.g., propensity to take risks), and processing of visual information are discussed.

Software developers come preloaded with overlearned behaviors, derived from the native culture of their formative years, and at least one human language, which they have spent many hours reading.

**Cognitive capitalism** Introductory economic issues are discussed.

The approach to software economics is from the perspective of the software engineer or software company rather than the perspective of the customer or user of software (which differs from much existing work, which adopts the customer or user perspective). The economics of products produced by intellectual effort has been growing in importance for some time (the chapter title provides a direction for how to think about the economic material); current economic theory and practice is predominantly based around the use of labor as the means of production;

**Ecosystems** Software is created in a development ecosystem that has been built and maintained by many people, and is used in a customer ecosystem that has usually been built by and maintained by many people. Once written software is unchanging, if the world in which the software is used changes, there will be pressure to change the software; commercial software systems change because the world in which they are used changes, and an incentive exists to make an investment in changing the software.

Various components of developer (e.g., careers), and development (e.g., APIs) ecosystems are discussed, along with non-software issues having an impact on developers.

**Projects** The incentive structures involved in building a software system, from bidding to delivery, and ongoing maintenance; risks and returns. Client/vendor interaction, management/developer/tester pulling in different directions, and what's in it for me.

**Reliability** What is the minimum viable reliability, i.e., what will the market tolerate? Coding mistakes, their lifetime, and the cost-effectiveness of reducing them. Likely operation profiles, and the likelihood of input values causing a fault to be experienced.

Testing to gain confidence that system reliability is within desired limits.

**Source code** Patterns of usage, how they relate to developer habits, application characteristics and language features; how might they be gainfully taken advantage of.

**Stories told by data** There is no point analysing data unless the results can be effectively communicated to the intended audience. Illustrations of techniques for communication a story found in data, along with issues to look out for in stories told by others.

**Statistics for software engineering** Developers are casual users of statistics and don't want to spend time learning lots of mathematics; they want to make use of techniques, not implement them. The approach used is similar to that of a Doctor examining a patient for symptoms that might suggest underlying processes.

It is assumed that readers have basic algebra skills and can interpret graphs; no other statistical knowledge is assumed on the part of the reader,

In many practical situations, the most useful expertise to have is knowledge of the application domain that generated the data and how the results might be applied. It is better to calculate an approximate answer to the correct problem than an exact answer to the wrong problem.

Because evidence-based software engineering has only recently started to be applied, there is uncertainty about which statistical techniques are most likely to be generally applicable. Therefore, an all encompassing approach is taken, and a broad spectrum of topics is covered, including:

- probability: an overview of basic principles, along with commonly used constructs, e.g., probability distributions, mean and variance, Markov chains, and rules of thumb,

- statistical concepts: sampling, describing data, p-value, confidence intervals, effect size, statistical power, bootstrapping, model building, comparing two or more groups,
- regression modeling: fitting data to an equation can provide information about which variables have the most power to explain measurable behavior. Software engineering measurements come in a wide variety of forms, while ordinary least-squares might be widely used in the social sciences, it is not always suitable for modeling the datasets encountered in software engineering; more powerful techniques such as generalized least-squares, nonlinear models, mixed models, additive models, structural equation models and others sometimes have to be used,
- time series: analysis of data where measurements at time  $t$  are correlated with measurements made at past times, e.g.,  $t - 1$ , is the domain of time series analysis (regression modeling assumes that successive measurements are independent of each other),
- survival analysis: measurements of time to an event occurring (e.g., death) are the domain of survival analysis,
- machine learning: various techniques for finding patterns in data when the person doing the analysis has no knowledge of the data, or the application domain that produced it. Sometimes we are clueless button pushers and machine learning can be a useful guide.

**Experiments** Probably the most common experiment performed by developers is hardware and software benchmarking. The many difficulties and complications involved in performing reliable benchmarks are illustrated.

General issues involving the design of experiments are discussed.

Surveys: Analysis of data obtained by asking people questions.

**Data cleaning** Garbage in garbage out. Data cleaning is the penultimate chapter (going unmentioned in some books), and is often the most time-consuming part of data analysis and a very necessary activity for obtaining reliable results.

Common data cleaning tasks, along with possible techniques for detecting potential problems and solving them using R are discussed.

**Overview of R** An overview of R aimed at developers who are fluent in at least one other computer language. The discussion concentrates on those language features likely to be commonly used, but behave surprisingly differently from languages the reader is likely to be familiar with.

**Obtaining and installing R:** If you cannot figure out how to obtain and install R, this book is not for you.

RStudio is a widely used R IDE that is also sometimes used by your author.

### 1.4.1 Why use R?

The main reasons for selecting R as the language+support library in which to write the statistical analysis programs in this book were:

- it is possible to quickly write a short program that solves the kind of problems that often occur when analysing software engineering data. The process often follows the sequence: read data from one of a wide variety of sources, operate on it using functions selected from a huge library of existing packages and finally graphically displaying the results or printing values,
- lots of people are using it: there is a very active ecosystem with many R books, active discussion forums where examples can be found, answers to old questions read and new questions posted,
- accuracy and reliability: a comparison of the reliability of 10 statistical software packages<sup>1178</sup> found that GAUSS, LIMDEP, Mathematica, MATLAB, R, SAS, and Stata provided consistent reliable estimation results, and a comparison of the statistical functions in Octave, Python, and R<sup>28</sup> found that R yielded the best results. A study<sup>29</sup> of the precision of five spreadsheets (Calc, Excel, Gnumeric, NeoOffice and Oleo) running under Windows

Vista, Ubuntu and Mac OS found that no one spreadsheet provided consistently good results (it was recommended that none of these spreadsheets be used for nonlinear regression and/or Monte Carlo simulation); another study<sup>1042</sup> found significant errors in the accuracy of the statistical procedures in various versions of Microsoft Excel.

- an extensive library of add-on packages (over 7,000 at the time of writing): CRAN, the Comprehensive R Archive Network is the official package library; some packages are only available on R-Forge and Github.

A freely available open source implementation is always nice to have.

## 1.5 Terminology, concepts and notation

Much of the basic terminology used in probability and statistics in common use today derives from gambling and experimental research in medicine and agriculture, because these were the domains where researchers working in the early days of statistics were employed.

- *group*: each sample is sometimes referred to as a group,
- *treatment*: The operation or process performed is often referred to as a treatment.
- *response variable* also known as a *dependent variable*, responds/depends to/on changes of values of explanatory variables; their behavior depends on the variables that are independent (at least of them),
- *explanatory variables* (also known as *independent, stimulus, predictor variables*) is used when the variables are used to make predictions, *control variables* is sometimes used in an experimental setting), are used to explain, predict or stimulate the value of response variables; they are independent of the response variable,
- Data is *truncated* when values below, or above some threshold are unobserved (or removed from the dataset).
- Data is *censored* when values below, or above some threshold are set equal to the threshold,
- *between subjects*: when samples are obtained from different groups of subjects, often with the different groups performing a task under different experimental conditions,
- *within subjects*: Comparing two or more samples obtained using the same group of subjects, often with the different subjects performing a task under two or more different experimental conditions,
- a *parametric test* is a statistical technique that assumes the sample has some known distribution,
- a *nonparametric test* is a statistical technique that does not make any assumptions about the distribution of the sample (the term *distribution free test* is sometimes used).

Some commonly encountered symbols and notation:

- $n!$  ( $n$  factorial), denotes the expression  $n(n - 1)(n - 2) \cdots 1$ ; calculated in the **factorial** function,
- $\binom{n}{r} = \frac{n!}{r!(n-r)!}$ , a commonly occurring quantity in the analysis of probability problems; calculated by the **choose** function,
- a hat above a variable,  $\hat{y}$ , denotes an estimate, in this case an estimate of  $y$ 's value,
- $\mu$  (mu), commonly denotes the mean value; calculated by the **mean** function,
- $\sigma$  (sigma), commonly denotes the standard deviation; calculated by the **sd** function. The terms 1-sigma, 2-sigma, etc. are sometimes used to refer to the probability of an event occurring. Figure 1.16 shows the sigma multiplier for various probabilities,
- $n \rightarrow \infty$  as  $n$  goes to infinity, i.e., becomes very very large,

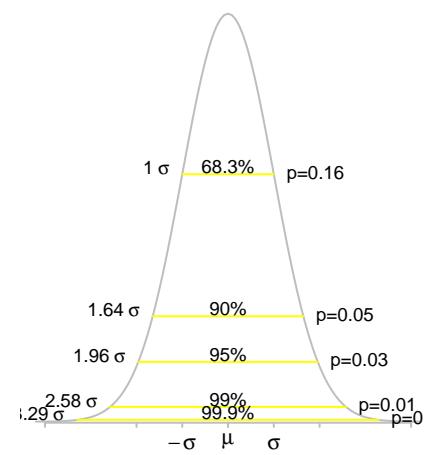


Figure 1.16: Normal distribution with total percentage of values enclosed within a given number of standard deviations. [code](#)

- $n \rightarrow 0$ , as  $n$  goes to zero, i.e., becomes very very small,
- $P(x)$ , the probability of  $x$  occurring and sometimes used to denote the Poisson distribution with parameter  $x$  (however, this case is usually written using  $\lambda$ , e.g.,  $P(\lambda)$ ),
- $P(a < X)$ , the probability that  $a < X$ . The functions `pnorm`, `pbinom` and `ppois` can be used to obtain the probability of encountering a value less than or equal to  $x$  for the respective distribution (e.g., Normal, Binomial and Poisson, as suggested by the naming convention),
- $P(|a - X|)$ , the probability of the absolute value of the difference between  $a$  and  $X$ ,
- $\prod_{i=1}^6 a_i$ , the product:  $a_1 \times a_2 \times \cdots \times a_6$ ,
- $\sum_{i=1}^6 P(a_i)$ , the sum:  $P(a_1) + P(a_2) + \cdots + P(a_6)$ ,

The sum the probabilities of all the mutually exclusive things that could happen, when an action occurs, is always one. For instance, when a die is rolled the six probabilities of a particular number occurring sum to one, irrespective of whether the die is fair or has been tampered with in some way.

- $P(D|S)$ , the probability of  $D$  occurring given that  $S$  is true; known as the *conditional probability*. For instance,  $S$  might be the event that two dice have been rolled, and their face-up numbers sum to five and  $D$  the event that the value of the first die is four.

The value can be calculated using the following:

$$P(D|S) = \frac{P(DS)}{P(S)}$$

where  $P(DS)$  is the probability of both  $D$  and  $S$  occurring together.

If  $D$  and  $S$  are independent of each other, then we have:

$$P(DS) = P(D)P(S)$$

and the above equation simplifies to:  $P(D|S) = P(DS)$

A lot of the existing theory in statistics assumes that variables are independent. Characteristics unique to each dice means using a different dice for each roll will produce values having slightly more independence than using the same dice twice.

**Convex/concave functions:** a function  $f(x)$  is *convex*, between  $a$  and  $b$ , if every chord of the function is above the function. If the chord is always below the function, it is concave; see Figure 1.17. The word convex tends to be spoken with a smiley face, while concave induces more of a frown.

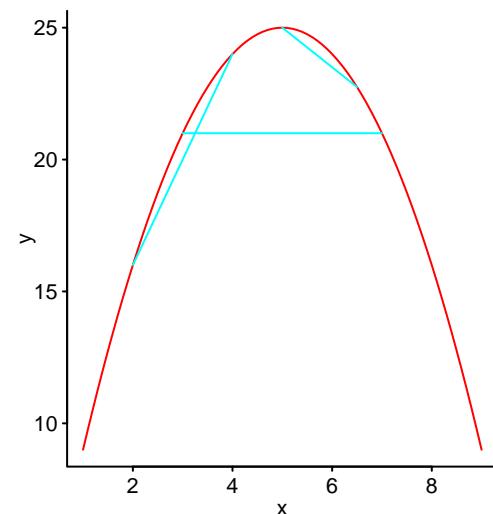
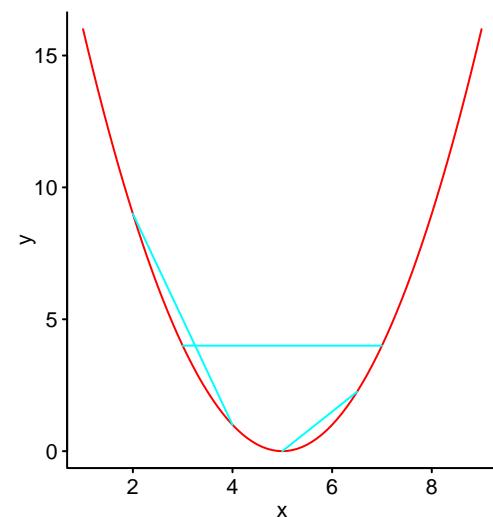


Figure 1.17: Example convex, upper, and concave, lower, functions; lines are three chords of the function. [code](#)

# Chapter 2

## Human cognitive characteristics

### 2.1 Introduction

Software systems are built and maintained by the creative output of human brains, which supply the cognitive effort that directs and performs the activities required; the operating characteristics of this computing machine are an essential component of any study of software engineering.

Modern humans evolved from earlier humanoids, who in turn evolved from earlier members of the ape family, who in turn evolved from etc., etc. The collection of cognitive characteristics present in the homo sapien brain is the end-result of a particular sequence of survival pressures that occurred over millions of years of evolutionary history; with the last common ancestor of the great apes, and the line leading to modern humans, living 5 to 7 million years ago,<sup>516</sup> the last few hundred thousand years spent as hunter-gatherers roaming the African savannah, followed by 10,000 years or so having a lifestyle that involved farming crops and raising domesticated animals.

Our skull houses a computing system that evolved to provide responses to problems that occurred in stone age ecosystems. However, this system is adaptable; neural circuits established for one purpose may be redeployed,<sup>45</sup> during normal development, for different uses, often without losing their original functions, e.g., in many people, learning to read and write involves repurposing the neurons in the ventral visual occipito-temporal cortex (an area of the brain involved in face processing and mirror-invariant visual recognition).<sup>384</sup> Reuse of neural circuitry is a central organizational principle of the brain.

The collection of cognitive characteristics supported by an animal's brain only makes sense in the context of the problems the species had to solve within the environment in which it evolved. Cognition and the environment are like the two blades of a pair of scissors (Figure 2.1), both blades have to mesh together to achieve the desired result.

- The structure of the natural environment places constraints on optimal performance (an approach to analyzing human behavior known as *rational analysis*),
- Cognitive, perception, and motor operations have their own sets of constraints (an approach known as *bounded cognition*, which targets good-enough performance).

Degraded, or incorrect, performance occurs when cognitive systems have to operate in a situation that violates the design assumptions of the environment they evolved to operate within; the assumptions are beneficial because they simplify the processing of ecologically common inputs. For instance, the human visual system assumes light shines from above, because it has evolved in an environment where this is generally true.<sup>709</sup> A consequence of this assumption of light shining from above is the optical illusion in Figure 2.2, that the top row appears as mounds while the lower row appears as depressions.

Optical illusions are accepted as curious anomalies of the eye/brain system; there is no rush to conclude that human eyesight is faulty. Failures of the cognitive system to produce answers in agreement with mathematical principles, chosen because they appeal to those making the selection, is a signal that the cognitive system has not been tuned by the environment in which

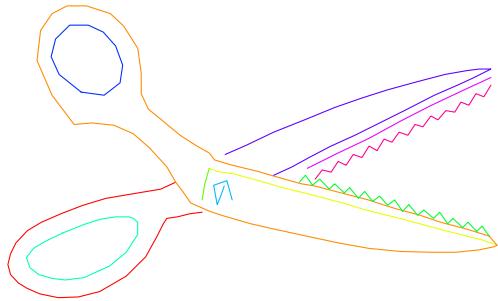


Figure 2.1: Unless cognition and the environment in which it operates closely mesh together, problems may be difficult or impossible to solve; the blades of a pair of scissors need to closely mesh for cutting to occur. [code](#)

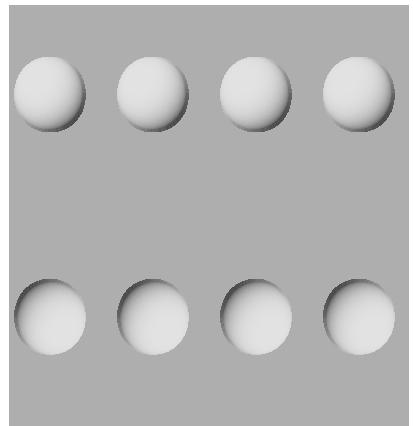


Figure 2.2: The assumption of light shining from above creates the appearance of bumps and pits. [code](#)

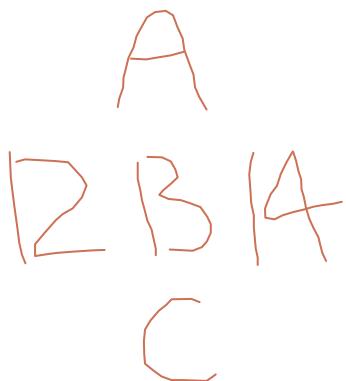


Figure 2.3: Overlearning enables readers to effortlessly switch between interpretations of curved lines. [code](#)

it has successfully operated to produce answers compatible with the selected mathematical principles.

This book is written from the viewpoint that the techniques used by people to produce software systems should to be fitted around the characteristics of the computing platform in our head (the view that developers should aspire to be omnipotent logicians is driven by human self-image, a counter-productive mindset).<sup>i</sup> Builders of bridges do not bemoan the lack of unbreakable materials available to them, they learned how to work within the limitations of the materials available.

Evolutionary psychology<sup>102, 106</sup> is an approach to psychology that uses knowledge and principles from evolutionary biology to help understand the operation of the human mind. Of course physical implementation details, the biology of the brain,<sup>811</sup> also have an impact on psychological performance.

The fact that particular cognitive abilities have benefits in some environments, that outweigh their costs, means they are to be found in a variety of creatures,<sup>131</sup> e.g., use of numbers by monkeys<sup>664</sup> and syntax by birds.<sup>1525</sup> A study by Mechner<sup>1057</sup> rewarded rats with food, if they pressed a lever  $N$  times (with  $N$  taking one of the values 4, 8, 12 or 16), followed by pressing a second lever. Figure 2.4 suggests that rat N1 is making use of an approximate number system. Other examples of non-human cognition are briefly discussed elsewhere, the intent is to show how deep-seated some of our cognitive abilities are, i.e., they need not be dependent on high-level functionality that is human specific.

Table 2.1 shows a division of human time scales by the kind of action that fits within each interval.

Scale (sec)	Time Units	System	World (theory)
10000000	months		Social Band
1000000			
100000			
10000	hours	Task	Rational Band
1000			
100			
10	10 sec	Unit task	Cognitive Band
1			
0.1			
0.01	10 msec	Neural circuit	Biological Band
0.001			
0.0001			

Table 2.1: Time scales of human action. Based on Newell.<sup>1148</sup>

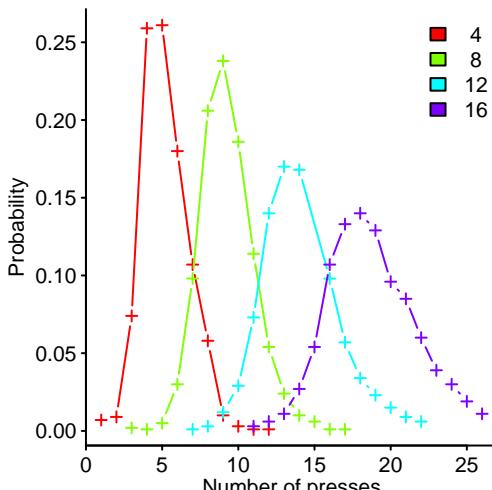


Figure 2.4: Probability that rat N1 will press a lever a given number of times before pressing a second lever to obtain food, when the target count is 4, 8, 12 and 16. Data extracted from Mechner.<sup>1057</sup> code

Does the brain contain a collection of modules (each handling particular functionality) or a general purpose processor? This question is the nature vs. nurture debate argument, rephrased using implementation details, i.e., a collection of modules pre-specified by nature or a general purpose processor that is configured by nurture. The term *modularity of mind* refers to a model of the brain<sup>515</sup> containing a general purpose processor attached to special purpose modules that handle perceptual processes (e.g., hearing and sight); the term *massive modularity hypothesis* refers to a model<sup>333</sup> that only contains modules.

Consciousness is the tip of the iceberg, most of what goes on in the mind is handled by the unconscious.<sup>278, 352, 1648</sup> Problems that are experienced as easy to solve, may actually involve very complex neural circuitry, and be very difficult to program computers to solve.

The only software engineering activities that could be said to be natural, in that prewired biological structures occur in the brain, involve social activities. The exactitude needed for coding is at odds with the fast and frugal approach of our unconscious mind,<sup>570</sup> whose decisions our conscious mind later does its best to justify.<sup>1648</sup> Reading and writing are not natural in the sense that specialist brain structures have evolved to perform these activities; it is the

<sup>i</sup> An analysis of the operation of human engineering suggests that attempting to modify our existing cognitive systems is a bad idea,<sup>186</sup> e.g., it is better to rewrite spaghetti code than try to patch it.

brain's generic ability to learn that enables this skill to be acquired through many years of deliberate practice.

What are the likely differences in cognitive performance between human males and females? A study by Strand, Deary and Smith<sup>1514</sup> analyzed Cognitive Abilities Test (CAT) scores from over 320,000 school pupils in the UK. Figure 2.5 provides a possible explanation for the prevalence of males at the very competent/incompetent ends of the scale and shows that women outnumber men in the middle competency band (over time differences have remained, but male/female performance ratios have changed<sup>1624</sup>). A model where one sex engages in relatively selective mate choice, produces greater variability in the opposite sex.<sup>700</sup>

A study by Jørgensen and Grimstad<sup>802</sup> asked subjects from three Asian and three east European countries to estimate the number of lines of code they wrote per hour, and the effort needed to implement a specified project (both as part of a project investigating cognitive biases). Both country and gender were significant predictors of the estimates made (see [developers/estimation-biases.R](#)).

While much has been written on how society exploits women, relatively little has been written on how society exploits men.<sup>118</sup> There are far fewer women than men directly involved in software engineering.<sup>ii</sup>

## 2.1.1 Modeling human cognition

Models of human cognitive performance are based on the results of experiments performed on subjects drawn almost entirely from Western, Educated, Industrialized, Rich and Democratic (WEIRD) societies.<sup>685</sup> While this is a problem for those seeking to uncover the cognitive characteristics of humans in general, it is not a problem for software engineering, because those involved have often had a WEIRD society education.

Characteristics of WEIRD people that appear to differ from the general population include:

- WEIRD people are willing to think about, and make inferences about, abstract situations, without the need for having had direct experience; studies<sup>990</sup> of non-WEIRD people have found they are unwilling to discuss situations where they don't have direct experience,
- when mapping numbers onto space, WEIRD people have been found to use a linear scale for mapping values between one and ten; studies of non-WEIRD people have found they often use a logarithmic scale,<sup>388</sup>
- WEIRD people have been found to have complex, but naive, models of the mechanisms that generate the everyday random events they observe.<sup>39</sup>

Much of the research carried out in cognitive psychology draws its samples from people between the ages of 18 and 21, studying some kind of psychology degree. There has been discussion<sup>100</sup> on the extent to which these results can be extended to the general populace, however, results obtained by sampling from this subpopulation are likely to be good enough for dealing with software engineering issues.

The reasons why students are not appropriate subjects to use in software engineering experiments, whose results are intended to be applied to professional software developers, are discussed in Chapter 13.

Several executable models of the operation of human cognitive processes have been created. The ACT-R model<sup>43</sup> has been applied to a wide range of problems, including learning, the visual interface, perception and action, cognitive arithmetic, and various deduction tasks.

Studies<sup>526</sup> have found a poor correlation between an individual's estimate of their own cognitive ability and measurements of their ability.

Studies of personnel selection<sup>1384</sup> have found that general mental ability is the best predictor of job performance.

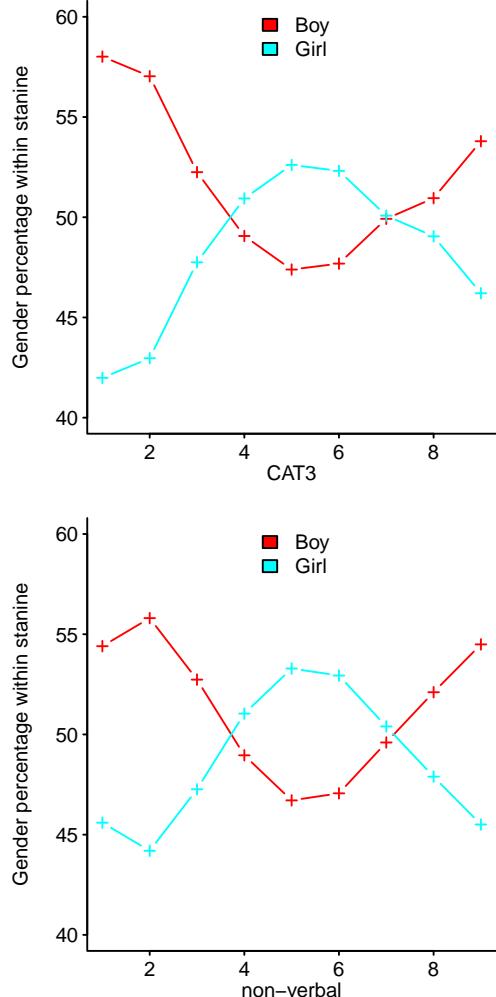


Figure 2.5: Boy/girl (aged 11-12 years) verbal reasoning, quantitative reasoning, non-verbal reasoning and mean CAT score over the three tests; each stanine band is 0.5 standard deviations wide. Data from Strand et al.<sup>1514</sup> [code](#)

<sup>ii</sup> When your author started working in software development, if there was a woman working on a team, his experience was that she would be at the very competent end of the scale (male/female ratio back then was what, 10/1?). These days, based on my limited experience, women are less likely to be as competent as they once were, but still a lot less likely, than men, to be completely incompetent; is the small number of incompetence women caused by a failure of equal opportunity regulations or because of a consequence of the small sample size?

Bayesian models of human cognition have been criticised for assuming that performance on a range of tasks is at, or near, optimal.<sup>191</sup> everyday human performance needs to be good enough, and an investment in finding an (near) optimal solution may not be cost effective.

The evidence-accumulation model is used in the cognitive analysis of decision-making. As its name suggests, the model operates by accumulating evidence until the quantity of evidence for one of the options exceeds the threshold needed to trigger its selection. The diffusion model,<sup>1310</sup> and the Linear Ballistic Accumulator (LBA) are currently two leading forms of this model.<sup>423</sup>

Figure 2.6 shows the basic components of evidence-accumulation models; this particular example is of a diffusion model, for a 2-choice decision (i.e., "A" or "B").

The *drift rate* is the average rate at which evidence accumulates in the direction of the correct response; noise in the process creates variability in the response time and the possibility of making an incorrect response. Evidence accumulation may start from a non-zero value.

The distance between the options' decision thresholds, on the evidence axis, impacts response time (increasing distance, increases response time), and error rate (decreasing distance, increases the likelihood that noise can cause sufficient evidence to accumulate to cross the threshold of the incorrect option).

The Linear Ballistic Accumulator model differs from the diffusion model, in that each option has its own register, which holds the evidence accumulated for that option; the first option whose accumulated evidence reaches threshold, is selected (the diffusion model accumulates evidence in a single register, until reaching the threshold of one of the options).

The measured response time includes what is known as *nondecision time*, e.g., tasks such as information encoding and time taken to execute a response.

Fitting data from experiments<sup>1309</sup> from a lexical decision task,<sup>iii</sup> to a diffusion model, shows that drift rate increases with word frequency, and that some characteristics of non-words (e.g., whether they were similar to words or were random characters) had an impact on the model parameters.

## 2.1.2 Embodied cognition

Embodied cognition is the theory that many, if not all, aspects of a person's cognitive processing are dependent on, or shaped by, sensory, motor, and emotional processes that are grounded in the features of their physical body.<sup>576</sup>

A study by Presson and Montello<sup>1283</sup> asked two groups of subjects to memorize the locations of objects in a room. Both groups were then blindfolded, and then asked to point to various objects; their performance was found to be reasonably fast and accurate. Subjects in one group were then asked to imagine rotating themselves 90°, they were then asked to point to various objects. Their performance was found to be much slower and less accurate. Subjects in the other group were asked to actually rotate 90°; while still blindfolded, they were then asked to point to various objects. The performance of these subjects was found to be as good as before they rotated. These results suggest that mentally keeping track of the locations of objects, a task that might be thought to be cognitive and divorced from the body, is in fact strongly affected by body position.

Tetris players have been found to prefer rotating an item on screen, as it descends, rather than mentally perform the rotation.<sup>853</sup>

A study by Shepard and Metzler<sup>1426</sup> showed subjects pairs of figures and asked if they were the same. Some pairs were different, while others were the same, but had been rotated relative to each other. The results showed a linear relationship between the angle of rotation (needed to verify that two objects were the same), and the time taken to make a matching comparison. Readers might like to try rotating, in their mind, the pair of images in each row of Figure 2.8, to find out if they are the same.

A related experiment by Kosslyn<sup>882</sup> showed subjects various pictures and asked questions about them. One picture was of a boat and subjects were asked a question about the front of the boat and then asked a question about the rear of the boat. The response time, when the question shifted from the front to the rear of the boat, was longer than when the question

<sup>iii</sup> Deciding whether a character string is a word.

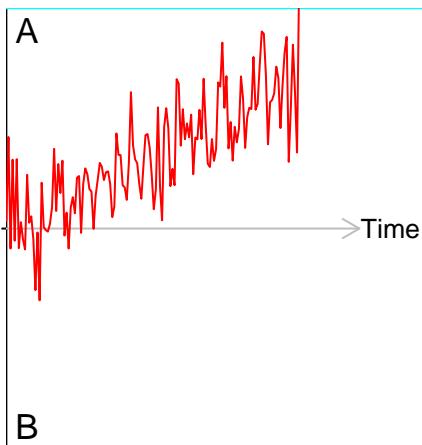


Figure 2.6: Example of the evolution of the accumulation of evidence for option "A", in a diffusion model. [code](#)

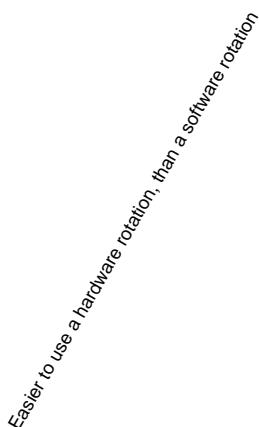


Figure 2.7: Rotating text in the real world; most easily done by tilting the head, or in the mind? [code](#)

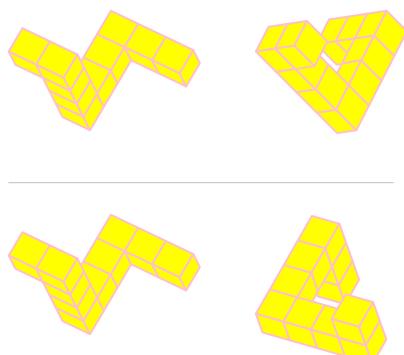


Figure 2.8: Two objects paired with another object that may be a rotated version. Based on Shepard et al.<sup>1426</sup> [code](#)

shifted from one about portholes to one about the rear. It was as-if subjects had to scan their image of the boat from one place to another to answer the questions.

Many WEIRD people use a mental left-to-right spatial orientation for the number line. This mental orientation has an embodiment in the *SNARC effect* (spatial numerical association of response codes). Studies<sup>386, 1171</sup> have shown subjects single digit values, and asked them to make an odd/even decision by pressing the left/right response button with the left-/right-hand; when using the right-hand response time decreases as the value increases (i.e., the value moves from left to right along the number line) and when using the left-hand response time decreases as the value decreases. The effect persists when arms are crossed, such that opposite hands are used for button pressing. Figure 2.9 shows the left/right-hand error rate found in a study of the SNARC effect.

### 2.1.3 Perfection is not cost-effective

Evolution is driven by survival of the fittest, i.e., it is based on relative performance; a consistent flawless performance is not only unnecessary, but a waste of precious resources.

Socially, making mistakes is an accepted fact of life and people are given opportunities to correct mistakes, if that is considered necessary.

For a given task, obtaining information on the kinds of mistakes that are likely to be made (e.g., entering numeric codes on a keyboard<sup>1201</sup>), and modeling the behavior (e.g., subtraction mistakes<sup>1595</sup> made by children learning arithmetic) is very time-consuming, even for simple tasks. Researchers are still working to build good models<sup>1470</sup> for the apparently simple task of text entry.

One technique for increasing the number of errors made by subjects, in an experiment, is to introduce factors that will increase the likelihood of mistakes being made. For instance, under normal circumstances the letters/digits viewed by developers are clearly visible, and the viewing time is not constrained; in experiments run under these conditions subjects make very few errors. To obtain enough data to calculate letter similarity/confusability, studies<sup>1112</sup> have to show subjects images of single letters/digits that have been visually degraded, or to limit the amount of time available to make a decision, or both, until a specified error rate is achieved.<sup>1557</sup> While such experiments may provide the only available information, on the topic of interest, their ecological validity has to be addressed (compared to say asking subjects to rate pairs of letters for similarity<sup>1447</sup>).

How often do people make mistakes?

A lower bound on human error rate, when performing over an extended period, is probably that of astronauts in space; making an error during a space mission can have very serious consequences, and there is a huge investment in astronaut training. NASA maintains several databases of errors made by operators during simulation training and actual missions; human error rates, for different missions, of between  $1.9 \cdot 10^{-3}$  and  $1.05 \cdot 10^{-4}$  have been recorded.<sup>264</sup>

Touch typists, who are performing purely data entry:<sup>1028</sup> with no error correction 4% (per keystroke), typing nonsense words (per word) 7.5%.

A number of human reliability analysis methods<sup>265</sup> for tasks in safety critical environments are available. The Cognitive Reliability Error Analysis Model (CREAM) is widely used; Calhoun et al<sup>241</sup> work through a calculation of the probability of an error during the International Space Station ingress procedure, using CREAM. The HEART method<sup>1669</sup> is another.

How do people respond when their mistakes are discovered?

A study by Jørgensen and Moløkken<sup>805</sup> interviewed employees, from one company, with estimation responsibilities. Analysis of the answers showed a strong tendency for people to perceive factors outside their control as important reasons for inaccurate estimates, while factors within their control were typically cited as reasons for accurate estimates.

## 2.2 Motivation

Motivation is a powerful influence on an individuals' priorities. The desire to complete an immediate task can cause a person to alter their priorities in a way they might not choose in

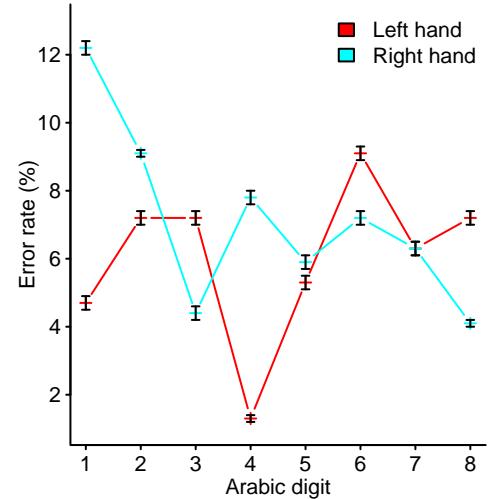


Figure 2.9: Error rate, with standard error, for the left/right hand from a study of the SNARC effect. Data from Nuerk et al.<sup>1171</sup> code

more relaxed circumstances. A study by Darley and Batson<sup>357</sup> asked subjects (theological seminary students) to walk across campus to deliver a sermon. Some subjects were told that they were late, and the audience was waiting, the remainder were not told this. Their journey took them past a victim moaning for help in a doorway. Only 10% of subjects who thought they were late stopped to help the victim, while 63% of the other subjects stopped to help. These results do not match the generally perceived behavior pattern of theological seminary students.

What motivations do developers have while working on software systems? Possible motivations include those that generally apply to people (e.g., minimizing cognitive effort, see Section 2.2.2), apply to company employees (e.g., seeking promotion, or doing a job because it pays for personal interests), apply to those involved in creative activities (e.g., organizing activities to maximize the pleasure obtained from work).

Hedonism is an approach to life that aims to maximise personal pleasure and happiness (see Section 11.3.5, for an analysis of a study investigating the importance of fun as a motivation for software development). Many theories of motivation take as their basis that people intrinsically seek pleasure and avoid pain, i.e., they are driven by *hedonic motivation*.

People involved in work that requires creativity can choose to include personal preferences and desires in their decision-making process, i.e., they are subject to hedonic motivation. To date, most of the research on hedonic motivation research has involved studies of consumer behavior.

The theory of *regulatory focus theory* is based around the idea that people's different approaches to pleasure and pain influences the approach they take towards achieving an end-state (or, end-goal). The theory contains two end states, one concerned with aspirations and accomplishments (a *promotion focus*), and the other concerned with attainment of responsibilities and safety (a *prevention focus*).

A promotion focus is sensitive to presence and positive outcomes, seeks to insure hits and insure against errors of omission. A prevention focus is sensitive to absence and negative outcomes, seeks to insure against correct rejections and insure against errors of commission.

People are able to exercise some degree of executive control over the priorities given to cognitive processes, e.g., deciding on speed/accuracy trade-offs. Studies<sup>520</sup> have found that subjects with a promotion focus will prefer to trade-off accuracy for speed of performance, and those with a prevention focus will trade-off speed for improved accuracy.

The concept of *Regulatory fit*<sup>696</sup> has been used to explain why people engage more strongly with some activities and "feel right" about it (because the activity sustains, rather than disrupts, their current motivational orientation or interests).

## 2.2.1 Built-in behaviors

Early researchers, investigating human behavior, found that people do not always respond ways that are consistent with the mathematically optimal models of behavior that had been created. The term *cognitive bias* has become the umbrella term used to describe such behavior.

Evolution selects for traits that provide an organism with advantages within its ecosystem; the failure of mathematical models, created by researchers, to predict human responses, is a failure of researchers to understand human behavior, not a failure of people to behave according to these models. Researchers are starting to create models<sup>960</sup> where, what were once thought to be cognitive biases, are actually ecologically rational uses of cognitive resources.

Evidence-based software engineering has to take into account whatever predispositions come included, as standard, with every human brain. The fact that many existing models of human behavior poorly describe real-life behaviors means that extreme caution is needed when attempting to model developer behaviors.

*Anchoring bias* and *confirmation bias* are two commonly discussed cognitive biases.

**Anchoring:** the behavior where people assign too much weight to the first piece of information they obtain, relating to the topic at hand.

A study by Jørgensen and Sjøberg<sup>807</sup> asked professionals and students to estimate the development effort for a project, with one group of subjects being given a low estimate from the

*customer*, a second group a low estimate (and no rationale), and the third group no *customer* estimate. The results (see [developer/anchor-estimate.R](#)) showed that estimates from subjects given a high/low customer estimate were much higher/lower than subjects who did not receive any customer estimate.

A study by Jørgensen and Grimstad<sup>802</sup> asked subjects to estimate the number of lines of code they wrote per hour, with subjects randomly split into two groups; one anchored with the question: “Did you on average write more or less than 1 Line of Code per work-hours in your last project?”, and the other with: “Did you on average write more or less than 200 Lines of Code per work-hours in your last project?” Fitting a regression model to the results showed the form of the question changed the mean estimate by around 72 lines (sd 10)... [developers/estimation-biases.R](#)

?

**Confirmation bias:** occurs when ambiguous evidence is interpreted as (incorrectly) confirming a person’s current beliefs about the world. For instance, developers interpreting program behavior as supporting their theory of how it operates, or using the faults exhibited by a program to conform their view that it was poorly written.

When shown data from a set of observations, a person might propose a set of rules that the processes generating the data adhere to. Given the opportunity to test proposed rules, what strategy are people likely to use?

A study by Wason,<sup>1639</sup> which became known as the *2–4–6 Task*, asked subjects to discover a rule known to the experimenter; subjects’ guessed a rule, told it to the experimenter, who told them whether the answer was correct. For instance, on being informed that the sequence 2–4–6 was an instance of the experimenter’s rule, possible subject rules might be “two added each time” or “even numbers in order of magnitude”, when perhaps the actual rule was “three numbers in increasing order of magnitude”.

An analysis of the rules created by subjects found that most were test cases designed to confirm a hypothesis about the rule (known as a *positive test strategy*), with few test cases attempting to disconfirm a hypothesis. Some subjects declared rules that were mathematically equivalent variations of rules they had already declared.

The use of a positive test strategy has been claimed to be a deficiency, because of the work of Popper,<sup>1263</sup> who proposed that scientists should perform experiments designed to disprove their hypothesis. Studies<sup>1375</sup> of the hypothesis testing strategies used by scientists found that positive testing is the dominant approach.

An analysis by Klayman and Ha<sup>861</sup> investigated the structure of the problem subjects were asked to solve. The problem is a search problem, find the experiment’s rule, and in some environments a positive test strategy is more effective for solving search problems, compared to a negative test strategy.

A positive test strategy is more effective when the sought after rule describes a minority case, i.e., there are more cases not covered by the rule, or when the hypothesized rule includes roughly as many cases as the actual rule, that is, the hypothesized rule is about the right size. Klayman and Ha claimed these conditions hold for many real-world rule search problems, and a positive test strategy is therefore an adaptive strategy; the real-worldness claim continues to be debated.<sup>1139</sup>

## 2.2.2 Cognitive effort

When attempting to solve a problem, a person’s cognitive system (consciously and unconsciously) makes cost/accuracy trade-offs. The details of how it forms an estimate of the value, cost and risk associated with an action, and carries out the trade-off analysis is not known (various models have been proposed<sup>619</sup>). An example of the effects of these trade-offs is provided by a study by Fu and Gray,<sup>528</sup> where subjects had to copy a pattern of colored blocks (on a computer-generated display). Remembering the color of the block to be copied, and its position in the target pattern, created a memory effort; a perceptual-motor effort was introduced by graying out the various areas of the display, where the colored blocks were visible; these grayed out areas could be made temporarily visible using various combinations of keystrokes and mouse movements. Subjects had the choice of expending memory effort (learning the locations of different colored blocks) or perceptual-motor effort (using

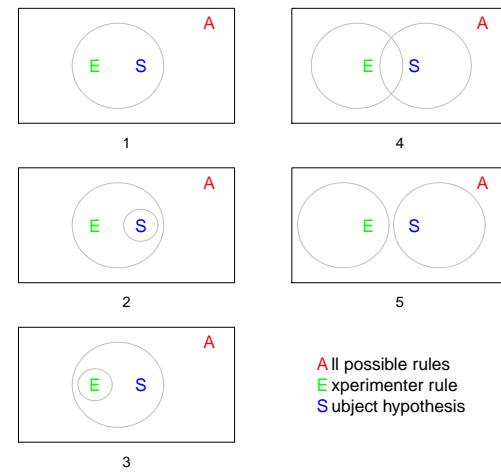


Figure 2.10: The five possible ways in which experimenter’s rule and subject’s rule hypothesis can overlap, in the space of all possible rules; based on Klayman et al.<sup>861</sup> code

keystrokes and mouse movements to uncover different areas of the display). A subject's total effort is the sum of the perceptual motor effort and the memory storage and recall effort.

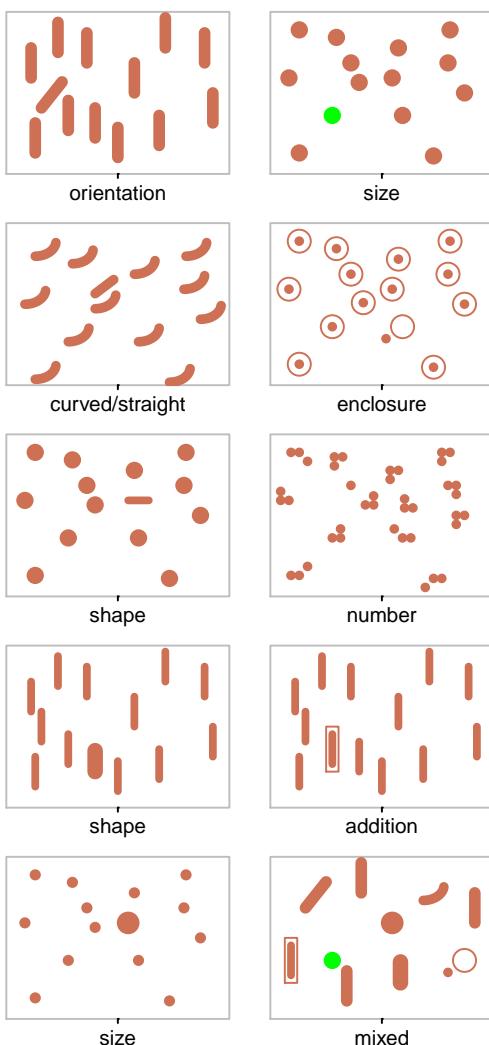
The subjects were split into three groups; one group had to expend a low effort to uncover the grayed out areas, the second acted as a control, and the third had to expend a high effort to uncover the grayed out areas. The results showed that the subjects who had to expend a high perceptual-motor effort, uncovered grayed out area fewer times than the other two groups. These subjects also spent longer looking at the areas uncovered, and moved more colored blocks between uncoverings. The subjects faced with a high perceptual-motor effort reduced their total effort by investing in memory effort. Another consequence of this switch of effort investment, to use of memory, was an increase in errors made.

Impact of incentives on cognitive performance...

Cognitive load theory...

What are the physical processes that generate the feeling of mental effort? The processes involved remain poorly understood,<sup>1424</sup> proposals include: metabolic constraints (the brain accounts for around 20% of heart output, and between 20% to 25% of oxygen and glucose requirements), but the energy consumption of the visual areas of the brain while watching television are higher than the consumption levels of those parts of the brain associated with difficult thinking, and that feeling of mental effort is the body's reaction to concentrated thinking is to try to conserve energy, for use in other opportunities that could arise in the immediate future, by creating a sense of effort.<sup>899</sup>

### 2.2.3 Attention



Most of the sensory information received by the brain does not require conscious attention, and can be handled by the unconscious. Conscious attention is like a spotlight shining cognitive resources on a chosen area. In today's world, there is often significantly more information available to a person than they have available attention resources, WEIRD people live in an attention economy.

People can direct attention to their internal thought processes and memories. For instance, read the bold text in the following paragraph:

Somewhere **Among** hidden **the** in **most** the **spectacular** Rocky Mountains **cognitive** near **abilities** Central City is Colorado **the** an **ability** old **to** miner **select** hid **one** a **message** box **from** of **another**. gold. **We** Although **do** several **this** hundred **by** people **focusing** have **our** looked **attention** for **on** it, **certain** they **cue**s have **such** not **as** found **type** it **style**.

What do you remember from the non-bold text? Being able to make a decision to direct conscious attention to inputs matching a given pattern is a technique for making efficient use of limited cognitive resources.

Much of the psychology research on attention has investigated how inputs from our various senses are handled. It is known that they operate in parallel, and at some point there is a serial bottleneck, beyond which point it is not possible to continue processing input stimuli in parallel. The point at which this bottleneck occurs is a continuing subject of debate; there are early selection theories, late selection theories, and theories that combine the two.<sup>1216</sup>

## 2.3 Visual processing

A study by Rogers and Monsell<sup>1345</sup> used the two tasks of classifying a letter as a consonant or vowel, and classifying a digit as odd or even. The subjects were split into three groups. One group was given the letter classification task, the second group the digit classification task, and the third group had to alternate (various combinations were used) between letter, and digit classification. The results showed that having to alternate tasks slowed the response times by 200 to 250 ms and the error rates went up from 2% to 3% to 6.5% to 7.5%. A study by Altmann<sup>32</sup> found that when the new task shared many features in common with the previous task (e.g., switching from classifying numbers as odd or even, to classifying them as less than or greater than five) the memories for the related tasks interfered, causing a reduction in subject reaction time, and an increase in error rate.

and this relates to software engineering...

Figure 2.11: Examples of features that may be preattentively processed; when items having distinct features are mixed together, individual items no longer jump out. Based on example in Ware.<sup>1637</sup> code

Visual processing is of interest to software development because it consumes cognitive resources and is a source of human input error. The 2-D image falling on the retina does not contain enough information to build the 3-D model we *see*; the mind creates this model by making assumptions about how objects in our environment move and interact.<sup>709</sup>

The perceptual systems of organisms have evolved to detect information in the environment that is relevant to survival, and ignore the rest. The relevant information is about opportunities *afforded* by the world.

The human visual system contains various hardware systems dedicated to processing visual information; low level details are preprocessed to build structures having a higher level of abstraction, e.g., lines. Preattentive processing, so called because it occurs before conscious attention,<sup>1295</sup> is automatic and apparently effortless. This preprocessing causes items to appear to *pop-out* from their surroundings. Figure 2.11 contains examples of items that pop-out at the reader.

Preattentive processing is independent of the number of distractors; a search for the feature takes the same amount of time whether it occurs with one, five, ten, or more other distractors. However, it is only effective when the features being searched for are relatively rare. When a display contains many, distinct features (the mixed category in Figure 2.11), the *pop-out* effect does not occur.

The *Gestalt laws of perception* (“gestalt” means “pattern” in German, also known as the *laws of perceptual organization*)<sup>1623</sup> are based on the underlying idea that the whole is different from the sum of its parts. These so-called *laws* do not have the rigour expected of a scientific law, and ought to be called by some other term (e.g., principle). The Gestalt principles are preprogrammed (i.e., there is no conscious cognitive cost). The following are some commonly occurring principles:

- continuity, also known as *good continuation*: Lines and edges that can be seen as smooth and continuous are perceptually grouped together, see upper plot in Figure 2.12,
- closure: elements that form a closed figure are perceptually grouped together, see upper plot in Figure 2.12,
- symmetry: treating two, mirror image lines as though they form the outline of an object, see second down plot in Figure 2.12. This effect can also occur for parallel lines,
- proximity: elements that are close together are perceptually grouped together, see second up plot in Figure 2.12,
- similarity: elements that share a common attribute can be perceptually grouped together, see lower plot Figure 2.12,
- other: principles include grouping by connectedness, grouping by common region, and synchrony.<sup>1209</sup>

Visually grouping the elements in a display, using these principles, is a common human trait. However, different people can make different choices, when perceptually grouping of the same collection of items. Figure 2.13 shows items on a horizontal line, which readers may group by shape, color, or relative proximity. A study by Kubovy and van den Berg<sup>889</sup> created a model that calculated the probability of a particular perceptual grouping (i.e., shape, color, or proximity in two dimensions) being selected for a given set of items.

Studies<sup>912</sup> have found that when mapping the prose specification of a mathematical relationship to a formula, the visual proximity of the applicable words has an impact on the error rate.

A number of studies have found that people are more likely to notice the presence of a distinguishing feature, than the absence of a distinguishing feature. This characteristic affects performance when searching for an item, when it occurs among visually similar items; it can affect reading performance&emdash;for instance, substituting an e for a c is more likely to be noticed than substituting a c for an e.

A study by Treisman and Souther<sup>1560</sup> found that visual searches were performed in parallel, when the target included a unique feature (i.e., search time was not affected by the number of background items), but were performed serially when the target had a unique feature missing (i.e., search time was proportional to the number of background items).

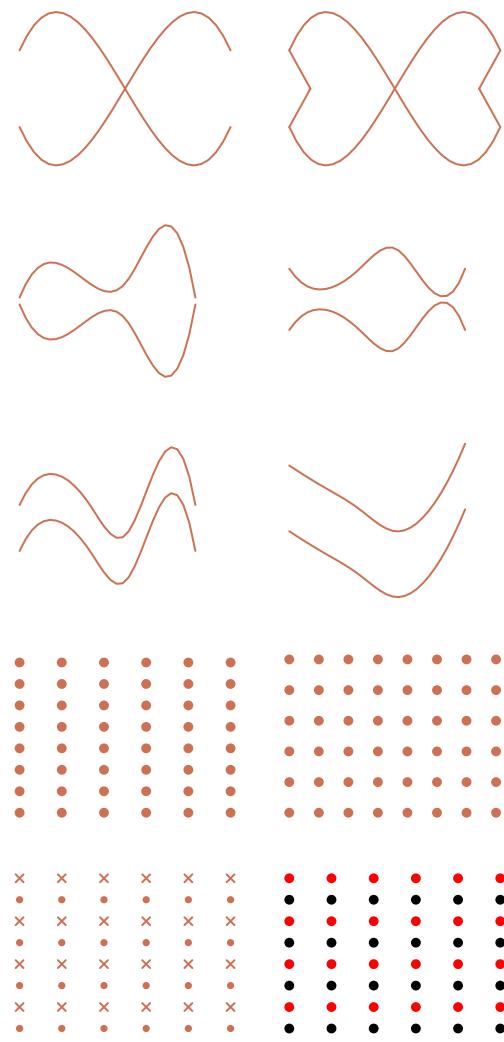


Figure 2.12: Continuity—upper left plot is perceived as two curved lines; Closure—when the two perceived lines are joined at their end (upper right), the perception changes to one of two cone-shaped objects; Symmetry and parallelism—where the direction taken by one line follows the same pattern of behavior as another line; Proximity—the horizontal distance between the dots in the lower left plot is less than the vertical distance, causing them to be perceptually grouped into lines (the relative distances are reversed in the right plot); Similarity—a variety of dimensions along which visual items can differ sufficiently to cause them to be perceived as being distinct; rotating two line segments by 180° does not create as big a perceived difference as rotating them by 45°; TODO look good. [code](#)

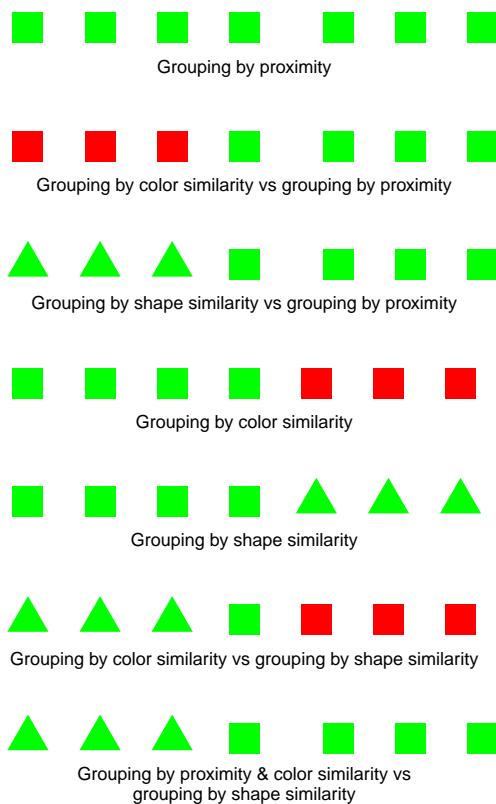


Figure 2.13: Perceived grouping of items on a line may be by shape, color or proximity. Based on kubovy et al.<sup>889</sup> code

### 2.3.1 Reading

Building software systems involves a significant amount of reading. Research on the cognitive processes involved in reading prose, written in human languages, has uncovered the basic processes and various models have been built.<sup>1314</sup>

When reading prose, a person's eyes make short rapid movements, known as *saccades*, taking 20 ms to 50 ms to complete; a saccade typically moves the eyes forward 6 to 9 characters. No visual information is extracted during a saccade, and readers are not consciously aware of them. Between saccades the eyes are stationary, typically for 200 ms to 250 ms, these stationary periods are known as *fixations*; a study of consumer eye movements,<sup>1248</sup> while comparing multiple brands, found a fixation duration of 354 ms when subjects were under high time pressure and 431 ms when under low time pressure.

Individual readers can exhibit considerable variations in performance, a saccade might move the eyes by one character, or 15 to 20 characters; fixations can be shorter than 100 ms or longer than 400 ms (there is also variation between languages<sup>1195</sup>). The content of the fixated text has a strong effect on performance.

The eyes do not always move forward during reading—10% to 15% of saccades move the eyes back to previous parts of the text. These backward movements, called *regressions*, are caused by problems with linguistic processing (e.g., incorrect syntactic analysis of a sentence), and oculomotor error (e.g., the eyes overshooting their intended target).

Saccades are necessary because the eyes' field of view is limited. Light entering an eye hits light-sensitive cells in the retina, where cells are not uniformly distributed. The visual field (on the retina) can be divided into three regions: foveal (the central 2°, measured from the front of the eye looking toward the retina), parafoveal (extending out to 5°), and peripheral (everything else). Letters become increasingly difficult to identify as their angular distance from the center of the fovea increases.

During the fixation period, two processes are active: identifying the word (or sequence of letters forming a partial word), and planning the next saccade, when to make it and where to move the eyes. Reading performance is speed limited, by the need to plan and perform saccades (removing the need to saccade, by presenting words at the same place on a display, results in a threefold speed increase in reading aloud, and a two-fold speed increase in silent reading). The time needed to plan and perform a saccade is approximately 180 ms —200 ms (known as the *saccade latency*), which means the decision to make a saccade occurs within the first 100 ms of a fixation.

The contents of the parafoveal region are partially processed during reading, and this increases a reader's perceptual span. When reading words written using alphabetic characters, the perceptual span extends from 3 to 4 characters on the left of fixation, to 14 to 15 letters to the right of fixation. This asymmetry in the perceptual span is a result of the direction of reading, attending to letters likely to occur next is a cost effective use of resources. Readers of Hebrew (which is read right-to-left) have a perceptual span that has opposite asymmetry (in bilingual Hebrew/English readers the direction of the asymmetry depends on the language being read, showing the importance of attention during reading).<sup>1313</sup>

Characteristics used by the writing system affect the asymmetry of the perceptual span and its width, e.g., the span can be smaller for Hebrew than English (Hebrew words can be written without the vowels, requiring greater effort to decode and plan the next saccade). It is also much smaller for writing systems that use ideographs, such as Japanese (approximately 6 characters to the right) and Chinese.

The perceptual span is not hardwired, but is attention-based. The span can become smaller when the fixated words become difficult to process. Also, readers extract more information in the direction of reading when the upcoming word is highly predictable (based on the preceding text).

Models of reading that have achieved some level of success include: Mr. Chips<sup>940</sup> is an ideal-observer model of reading (it is not intended to model how humans read, but to establish the pattern of performance, when optimal use is made of available information), which attempts to calculate the distance, in characters, of the next saccade; it combines information from visual data obtained by sampling the text through a retina, lexical knowledge of words and their relative frequencies, and motor knowledge of the statistical accuracy of saccades and uses the optimization principle of entropy minimization. The SWIFT model<sup>445</sup> attempts to provide a realistic model of saccade generation, while E-Z Reader<sup>1261</sup> attempts to account for how cognitive and lexical processes influence the eye movements of skilled readers (it can handle the complexities of *garden path* sentences<sup>1320iv</sup>).

English text is written left to right, on lines that go down the page. The order in which the components of a formula are read depends on its contents, with the visual processing of subexpressions by experienced users driven by the mathematical syntax,<sup>770,771</sup> with the extraction of syntax happening in parallel.<sup>1385</sup>

A study by Pelli, Burns, Farell, and Moore<sup>1229</sup> found that 2,000 to 4,000 trials were all that was needed for novice readers to reach the same level of efficiency as fluent readers in the letter-detection task (efficiency was measured by comparing human performance compared to an ideal observer). They tested subjects aged 3 to 68 with a range of different (and invented) alphabets (including Hebrew, Devanagari, Arabic and English). Even fifty years of reading experience, over a billion letters, did not improve the efficiency of letter detection. They also found this measure of efficiency was inversely proportional to letter perimetric complexity (defined as, inside and outside perimeter squared, divided by *ink* area).

Choice of display font is something that many developers are completely oblivious to. The use of Roman, rather than Helvetica (or serif vs. sans serif), is often claimed to increase reading speed and comprehension. The issues involved in selecting fonts are covered in a report detailing “Font Requirements for Next Generation Air Traffic Management Systems”.<sup>212</sup>

Vision provides information about what people are thinking about; gaze follows shifts of visual attention. Tracking a subject’s eye movements and fixations, when viewing images or real-life scenes, is an established technique in fields such as marketing and reading research; this technique is now starting to be used in research of developer code reading (see Figure 2.16).

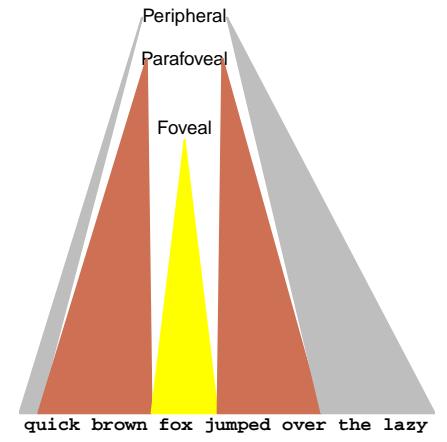


Figure 2.15: The foveal, parafoveal and peripheral vision regions when three characters visually subtend 3°. Based on Schotter et al.<sup>1392</sup> code

## 2.4 Memory systems

Memory evolved to supply useful, timely information to an organism’s decision-making systems.<sup>863</sup> Memory subsystems are scattered about the brain, with each subsystem/location believed to process and store distinct kinds of information. Figure 2.17 illustrates a current model of known long-term memory subsystems, along with the region of the brain where they are believed to operate.

Declarative memory has two components, each processing specific instances of information, one handles facts about the world, while the other, episodic memory, deals with events (i.e., the capacity to experience an event in the context in which it occurred; it is not known if non-human brains support episodic memory). We are consciously aware of declarative memory, facts and events can be consciously recalled; it is the kind of memory that is referred to in everyday usage as *memory*. Declarative memory is representational, it contains information that can be true or false.

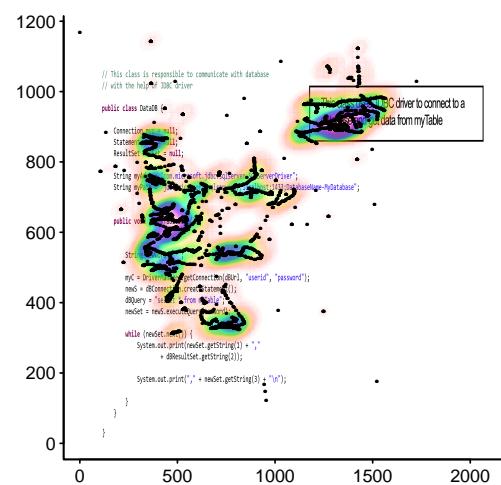


Figure 2.16: Heat map of one subject’s cumulative fixations (black dots) on a screen image. Data kindly provided by Ali.<sup>23</sup> code

<sup>iv</sup> In “Since Jay always jogs a mile seems like a short distance.” readers experience a disruption that is unrelated to the form or meaning of the individual words; the reader has been led down the syntactic garden path of initially parsing the sentence such that a *mile* is the object of *jogs* before realizing that a *mile* is the subject of *seems*.

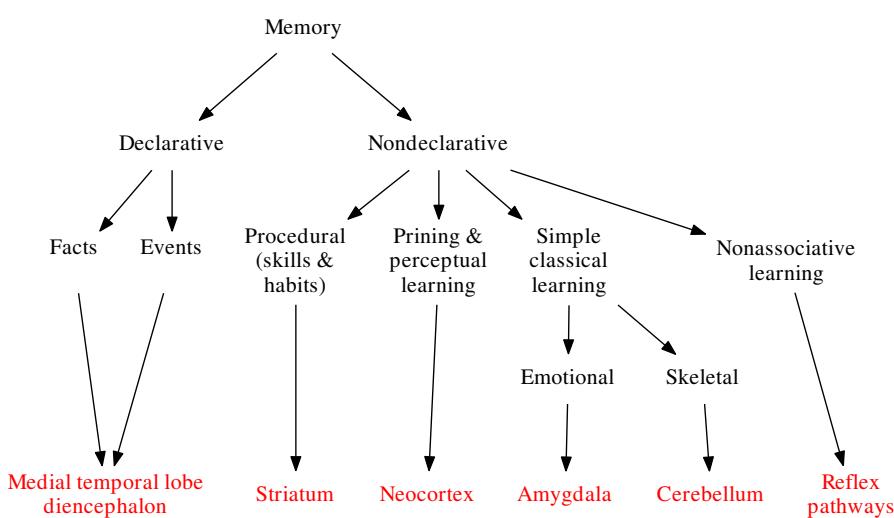


Figure 2.17: Structure of mammalian long-term memory subsystems; brain areas in red. Based on Squire et al.<sup>1482</sup>

Nondeclarative memory (also known as *implicit memory*) extracts information from recurring instances of an experience, to form skills (e.g., speaking a language) and habits, simple forms of conditioning, priming (response to a stimulus is modified by a preceding stimulus;<sup>1324</sup> an advantage in a slowly changing environment, where similar events are likely to occur on a regular basis), and perceptual learning (gradual improvement in the detection or discrimination of visual stimuli with practice).

Information in nondeclarative memory is extracted through unconscious performance, e.g., riding a bike; it is an unconscious memory and is not available for conscious recall (information use requires reactivation of the subsystem where the learning originally occurred).

These subsystems operate independently and in parallel, the parallelism creates the possibility of conflicting signals being fed as inputs to higher level systems. For instance, a sentence containing the word **blue** may be misinterpreted because information about the word, and the color in which it appears, **green**, is returned by different memory subsystems (known as the *Stroop effect*).

A Stroop-like effect has also been found to occur with lists of numbers. Readers might like to try counting the number of characters occurring in each row in the outside margin. The effort of counting the digit sequences is likely to be greater, and more error prone, than for the letter sequences.

Studies<sup>1221</sup> have found that when subjects are asked to enumerate visually presented digits, the amount of Stroop-like interference depends on the arithmetic difference between the magnitude of the digits used, and the quantity of those digits displayed. Thus, a short, for instance, list of large numbers is read more quickly, and with fewer errors, than a short list of small numbers. Alternatively a long list of small numbers (much smaller than the length of the list) is read more quickly and with fewer errors than a long list of numbers where the number has a similar magnitude to the length of the list.

Taking advantage of patterns that can be seen in the environment is one technique for reducing memory load.

Studies have found that people organize their memory for objects within their visual field, according to the relative positions of the objects. A study by McNamara, Hardy, and Hirtle<sup>1051</sup> gave subjects two minutes to memorize the location of objects on the floor of a room (e.g., upper plot of Figure 2.18). The objects were then placed in a box, and subjects were asked to replace the objects in their original position; the memorize/recall cycle was repeated, using the same layout, until the subject could place all objects in their correct position.

The order in which each subject recalled the location of objects was mapped to a hierarchical tree (one for each subject). The resulting trees (e.g., lower plot of Figure 2.18) showed how subjects' spatial memory of the objects had an organization based on the spatial distance between items.

Other research on the interaction between human memory and software development includes: a study by Altmann,<sup>31</sup> who built a computational process model, based on SOAR, and fitted it to 10.5 minutes of programmer activity (debugging within an emacs window); the simulation was used to study the memories, called near-term memory by Altmann, built up while trying to solve a problem.

thimble	cigarettes	radio	lightbulb	can	cufflink	lock	flashlight	book	cork	eraser	envelope	truck	cup	ruler	screw	egg	pen	button	ring	gun	ball	knife	scissors	shoe
3	3	3	3	a a a a a	8 8 8	z z			1 1	t t t t	glu	6 6 6 6 6												

Figure 2.18: Example object layout, and the corresponding ordered tree produced from the answers given by one subject. Data extracted from McNamara et al.<sup>1051</sup> code

### 2.4.1 Short term memory

As its name implies, *short term memory* (STM) is a memory system that can hold information for short periods of time. Short term memory is the popular term for what cognitive psychologists call *working memory*, named after its function, rather than the relative duration it can hold information. Early researchers explored its capacity, and a paper by Miller<sup>1083</sup> became the source cited for the now-famous  $7 \pm 2$  rule (Miller did not propose  $7 \pm 2$  as the capacity of STM; he simply drew attention to the fact that this range of values fitted the results of several experiments). Things have moved on in the 60+ years since the publication of his paper,<sup>784</sup> with benchmarks now available for evaluating models of STM.<sup>1176</sup>

Readers might like to try measuring their STM capacity, using the list of numbers in the outside margin. Any Chinese-speaking readers can try this exercise twice, using the English and Chinese words for the digits (use of Chinese should enable readers to apparently increase the capacity of STM). Slowly and steadily read the digits in a row, out loud. At the end of each row, close your eyes and try to repeat the sequence of digits in the same order. If you make a mistake, go on to the next row. The point at which you cannot correctly remember the digits in any two rows, of a given length, indicates your capacity limit—the number of digits in the previous two rows.

The performance impact of reduced working memory capacity can be shown by having people perform two tasks simultaneously. A study by Baddeley<sup>86</sup> measured the time taken to solve a simple reasoning task (e.g.,  $B \rightarrow A$ : “A follows B” True or False?), while remembering a sequence of digits (the number of digits is known as the *digit load*). Figure 2.19 shows response time (left axis) and percentage of incorrect answers (right axis) for various digit loads.

Measuring memory capacity using sequences of digits relies on a variety of assumptions, such as assuming all items consume the same amount of memory resources (e.g., digits and letters are interchangeable), that relative item ordering is implicitly included in the measurement and that individual concepts are the unit of storage. Subsequent studies have completely reworked models of STM. What the preceding exercise measured was the amount of *sound* that could be held STM. The sound used to represent digits in Chinese is shorter than in English, and using Chinese should enable readers to maintain information on more digits (average 9.9<sup>721</sup>) using the same amount of sound storage. A reader using a language in which the sound of the digits is longer, maintains information on fewer digits, e.g., average 5.8 in Welsh,<sup>443</sup> and the average for English is 6.6.

Observations of a  $7 \pm 2$  capacity limit derive from the number of English digits spoken in two seconds of sound<sup>88</sup> (people speak at different speeds, which is one source of variation included in the  $\pm 2$ ; an advantage for fast talkers). The two seconds estimate is based on the requirement to remember items, and their relative order; the contents of STM do not get erased after two seconds, this limit is the point at which degradation of its contents start to become noticeable.<sup>111</sup> If recall of item order is not relevant, then the limit increases because loss of this information is not relevant.

Studies<sup>1177</sup> involving multiple tasks have been used to distinguish the roles played by various components of working memory (e.g., storage, processing, supervision and coordination). Figure 2.20 shows the components believed to make up working memory, each with its own independent temporary storage areas, each holding and using information in different ways.

The central executive is assumed to be the system that handles attention, controlling the phonological loop, the visuo-spatial sketch pad, and the interface to long-term memory. The central executive needs to remember information while performing tasks, such as text comprehension and problem solving. It has been suggested that the focus of attention is capacity-limited, but that the other temporary storage areas are time-limited (without attention to rehearse them, they fade away).<sup>338</sup>

Visual information held in the visuo-spatial sketch pad decays very rapidly. Experiments have shown that people can recall four or five items immediately after they are presented with visual information, but that this recall rate drops very quickly after a few seconds.

Mental arithmetic provides an example of how different components of working memory can be combined to solve a problem that is difficult to achieve using just one component; multiply 23 by 15 without looking at this page. Information about the two numbers, and the intermediate results, has to be held in short term memory and the central executive. Now

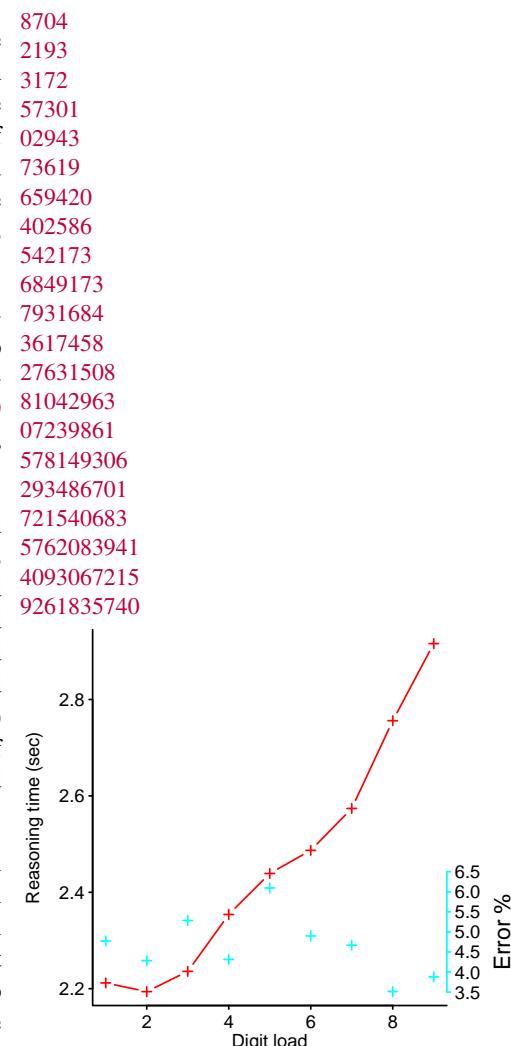


Figure 2.19: Response time (left axis) and error percent (right axis) on reasoning task with given number of digits held in memory. Data extracted from Baddeley.<sup>86</sup> code

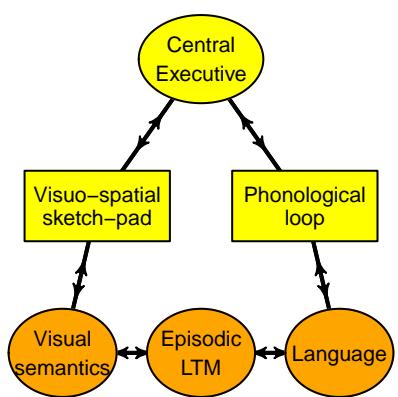


Figure 2.20: Major components of working memory: working memory in yellow, long-term memory in orange. Based on Baddeley.<sup>87</sup> code

perform another multiplication, but this time look at the two numbers being multiplied (see outer margin for values), while performing the multiplication.

- 26**  
**12** Looking at the numbers reduces the load on working memory (because cognitive effort does not have to be invested in remembering the numbers being multiplied).

Table 2.2 contains lists of words; those at the top of the table contain a single syllable, those at the bottom multiple syllables. Readers should have no problems remembering a sequence of five single-syllable words, a sequence of five multi-syllable words is likely to be more difficult. As before, read each word slowly out loud.

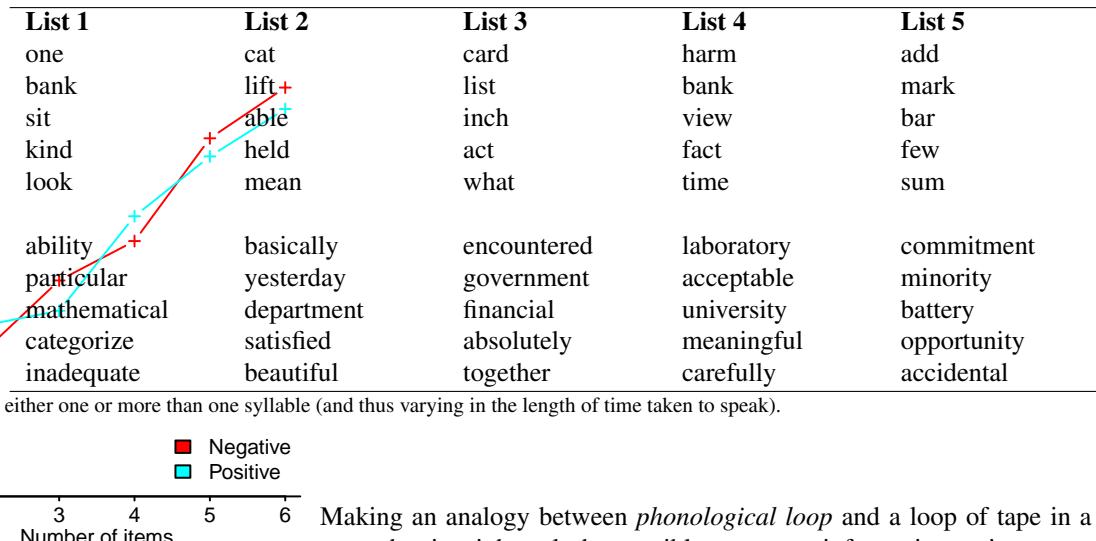


Figure 2.21: Yes/no response time (in milliseconds) as a function of the number of digits held in memory. Data extracted from Sternberg.<sup>1505</sup> [code](#)

Making an analogy between *phonological loop* and a loop of tape in a tape recorder, suggests that it might only be possible to extract information as it goes past a *read-out point*. A study by Sternberg<sup>1505</sup> asked subjects to hold a sequence of digits in memory, e.g., 4185, and measured the time taken to respond yes/no, about whether a particular digit was in the sequence. Figure 2.21 shows that as the number of digits increases, the time taken for subjects to respond increases; another result was that response time was not affected by whether the answer was yes or no. It might be expected that a yes answer would enable searching to terminate, but the behavior found suggests that all digits were always being compared. Different kinds of information has different search response times.<sup>256</sup>

Extrapolating the results from studies based on the use of natural language,<sup>360</sup> to the use of computer languages, need to take into account that reader performance has been found to differ between words (character sequences having a recognized use in the reader's native human language) and non-words, e.g., naming latency is lower for words,<sup>1647</sup> and more words can be held in short term memory<sup>736</sup> (i.e.,  $\text{word\_span} = 2.4 + 2.05 \times \text{speech\_rate}$ , and  $\text{nonword\_span} = 0.7 + 2.27 \times \text{speech\_rate}$ ).

The ability to comprehend syntactically complex sentences is correlated with working memory capacity.<sup>847</sup> A study by Miller and Isard<sup>1084</sup> investigated subjects' ability to memorize sentences that varied in their degree of embedding. The following sentences have increasing amounts of embedding (Figure 2.22 shows the parse-tree of two of them):

She liked the man that visited the jeweller that made the ring that won the prize that was given at the fair.

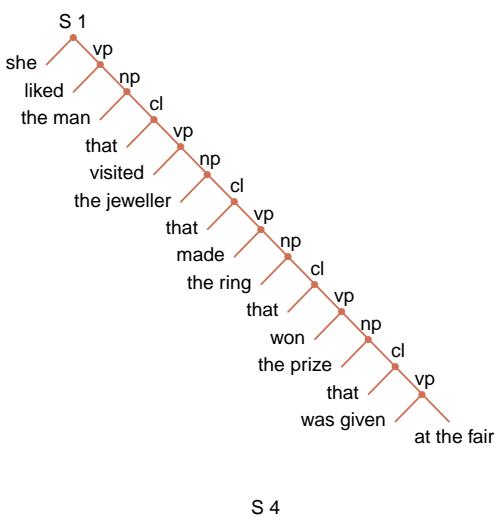
The man that she liked visited the jeweller that made the ring that won the prize that was given at the fair.

The jeweller that the man that she liked visited made the ring that won the prize that was given at the fair.

The ring that the jeweller that the man that she liked visited made won the prize that was given at the fair.

The prize that the ring that the jeweller that the man that she liked visited made won was given at the fair.

Subjects' ability to correctly recall wording, decreased as the amount of embedding increased, although performance did improve with practice. People have significant comprehension difficulties when the degree of embedding in a sentence exceeds two.<sup>172</sup>



Human language has a grammatical structure that enables it to be parsed serially (e.g., as it is spoken).<sup>1244</sup> One consequence of this expected characteristic of sentences, is that so called *garden path* sentences (where one or more words at the end of a sentence changes the parsing of words read earlier) generate confusion (requiring conscious effort to reason about what has been said):

The old train their dogs.

The patient persuaded the doctor that he was having trouble with to leave.

While Ron was sewing the sock fell on the floor.

Joe put the candy in the jar into my mouth.

The horse raced past the barn fell.

A study by Mathy, Chekaf and Cowan<sup>1027</sup> investigated the impact, on subject performance, of various patterns in a list of items to be remembered. The upper plot in Figure 2.23 shows an example of how items on a list might share one or more of the attributes: color, shape or size. A list was said to be “chunkable”, if its items shared attributes in a way that allowed subjects to reduce the amount of information needed to encode the items (e.g., all purple, small/large, triangle then circle); the items on a “non-chunkable” list did not share attributes in a way that reduced the amount of information that subjects needed to remember. A chunkability value was calculated for each list.

In the simple span task, subjects saw a list of items, and after a brief delay had to reconstruct the list. In the complex span task, subjects saw a list of items, but had to judge the correctness of a simple arithmetic expression, before reconstructing the list.

An item span score was calculated for subject, based on the number of items correctly recalled from each list they say (the chunkability of the list was included in the calculation). The lower plot in Figure 2.23 shows the mean span over all subjects, with standard deviation...

The two competing models, of loss of information in working memory, are:<sup>476</sup> passive decay (information fades away unless a person consciously spends time rehearsing or refreshing it), and loss through interference with new incoming information (a person has to consciously focus attention on particular information to prevent interference from new information).

## 2.4.2 Episodic memory

Episodic memory is memory for personally experienced events that are remembered as such, i.e., the ability to recollect specific events or episodes in our lives. When the remembered events occurred sometime ago, the term *autobiographical memory* is sometimes used.

What impact does the passage of time have on episodic memories?

A study by Altmann, Trafton and Hambrick<sup>33</sup> investigated the effects of interruption on a task involving seven steps. Subjects performed the same task 37 times, and were interrupted at random intervals during the 30-50 minutes it took to complete the session. Interruptions required subjects to perform a simple typing task that took, on average, 2.8, 13, 22 or 32 seconds. Figure 2.24 shows the percentage of sequencing errors made immediately after an interruption, and under normal working conditions (in red; sequence error rate without interruptions in green). A sequence error occurs when an incorrect step is performed, e.g., step 5 is performed again, after performing step 5, when step 6 should have been performed; the offset on the x-axis is the difference between the step performed, and the one that should have been performed; the sequence error rate, as a percentage of the total number of tasks performed at each interruption interval, was 2.4, 3.6, 4.6 and 5.1%. The lines are predictions made by a model fitted to the data.

## 2.4.3 Recognition and recall

Recognition memory is the ability to recognise previously encountered items, events or information. Studies<sup>877</sup> have found that people can often make a reasonably accurate judgement about whether they know a piece of information or not, even if they are unable to recall that information at a particular instant; the so-called *feeling of knowing* is a good predictor of subsequent recall of information,

Recall memory is the ability to retrieve previously encountered items, events or information. Studies of recall memory have investigated factors that influence recall performance (e.g.,

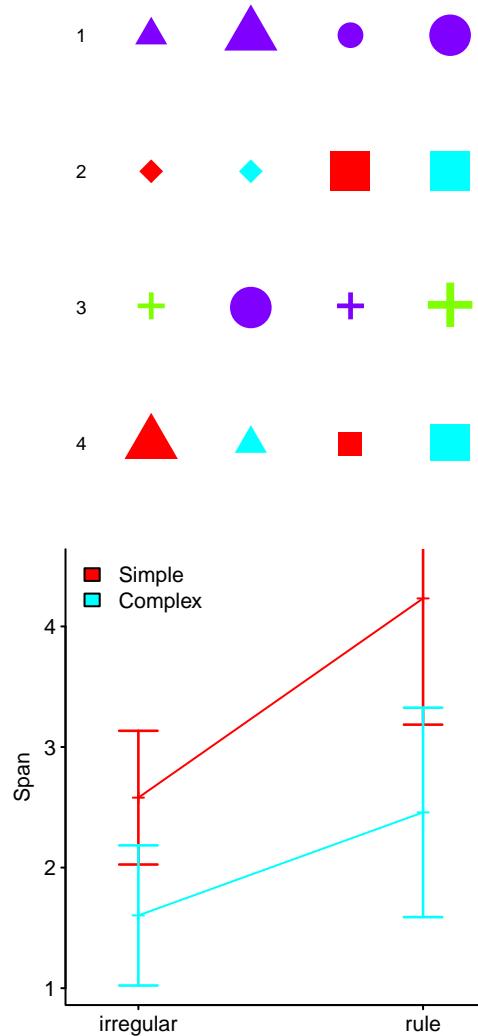


Figure 2.23: Examples of the kind of pattern of symbol sequence stimuli used by Mathy et al.<sup>1027</sup> code

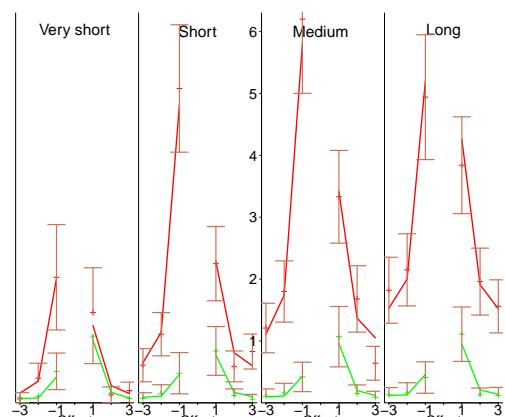


Figure 2.24: Sequencing errors (as percentage), after interruptions of various length (red), including 95% confidence intervals, sequence error rate without interruptions in green; lines are fitted model predictions. Data from Altmann et al.<sup>33</sup> code

the structure of the information to be remembered, associations between new and previously learned information, and the interval between learning and recall), techniques for improving recall performance, and how information might be organized in memory. Various models of recall memory have been proposed<sup>220</sup>...

The environment in which information was learned can have an impact on recall performance. A study by Godden and Baddeley<sup>578</sup> investigated subjects' recall of words memorized in two different environments. Subjects were divers, who learned a list of spoken words, either while submerged underwater wearing scuba apparatus, or while sitting at a table on dry land. The results found that subjects recall performance was significantly better, when performed in the environment in which the word list was learned.

When asked to retrieve members of a category, people tend to produce a list of semantically related items, before switching to list another cluster of semantically related items and so on. This pattern of retrieval is similar to that seen in strategies of optimal food foraging,<sup>701</sup> however the same behavior can also emerge from a random walk on a semantic network built from human word-association data.<sup>2</sup>

Chunking is a technique commonly used by people to help them remember information. A chunk is a small set of items ( $4\pm1$  is seen in many studies) having a common, strong, association with each other (and a much weaker one to items in other chunks). For instance, Wickelgren<sup>1660</sup> found that people's recall of telephone numbers is optimal, if numbers are grouped into chunks of three digits. An example, using random-letter sequences is: **fbi**cb. The trigrams (fbi, cbs, ibm, irs), within this sequence of 12 letters, are well-known acronyms. A person who notices this association can use it to aid recall. Several theoretical analyses of memory organizations have shown that chunking of items improves search efficiency (optimal chunk size 3–4,<sup>407</sup> number items at which chunking becomes more efficient than a single list, 5–7<sup>999</sup>).****

A study by Klahr, Chase, and Lovelace<sup>860</sup> investigated how subjects stored letters of the alphabet in memory. Through a series of time-to-respond measurements, where subjects were asked to name the letter that appeared immediately before or after the presented probe letter, they proposed the alphabet-storage structure shown in Figure 2.25.

#### 2.4.4 Serial order information

Information about the world is processed serially. Studies<sup>743</sup> have consistently found a variety of patterns in recall of serial information, such as a list of recently remembered items; patterns include:

- higher probability of recall for items at the start (the *primacy effect*) and end (the *recency effect*) of a list (known as the *serial position effect*,<sup>1119</sup> see Figure 2.26). The probability that an item will be remembered as occupying position  $i$ , at time  $t+1$ , is approximately given by:<sup>1133</sup>  $P_{i,t+1} = (1-\theta)P_{i,t} + \frac{\theta}{2}P_{i-1,t} + \frac{\theta}{2}P_{i+1,t}$ , for interior positions, and  $P_{1,t+1} = (1 - \frac{\theta}{2})P_{1,t} + \frac{\theta}{2}P_{2,t}$ , for the first item (and similarly for the last item). One study,<sup>1133</sup> involving lists of five words, found that applying a value of  $\theta = 0.12$ , for each 2-hour retention interval, produced predictions in reasonable agreement with the experimental data (more accurate models are available<sup>220</sup>),
- recall of a short list tends to start with the first item, and progress in order through the list, while for a long list people are more likely to start with one of the last four items;<sup>1635</sup> when prompted by an entry on the list people are most likely to recall the item following it<sup>727</sup> (see [developers/misc/HKJEP99.R](#)),
- when recalling items from an ordered list, people tend to make anticipation errors (i.e., recall a later item early), shifting displaced items further along the list.<sup>475</sup>

A study by Adelson<sup>8</sup> investigated the organization present in subject's recall order of previously remembered lines of code. Subjects saw a total of 16 lines of code, one line at a time, and were asked to memorise each line for later recall. The lines were taken from three simple programs (two containing five lines each and one containing six lines), the order being randomised for each subject. Five subjects were students who had recently taken a course using the language used, and five subjects had recently taught a course using the language.

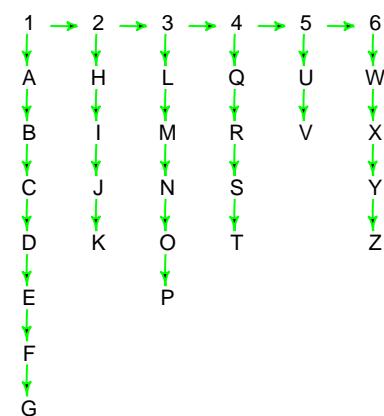


Figure 2.25: Semantic memory representation of alphabetic letters (the numbers listed along the top are place markers and are not stored in subject memory). Readers may recognize the structure of a nursery rhyme in the letter sequences. Derived from Klahr.<sup>860</sup> [code](#)

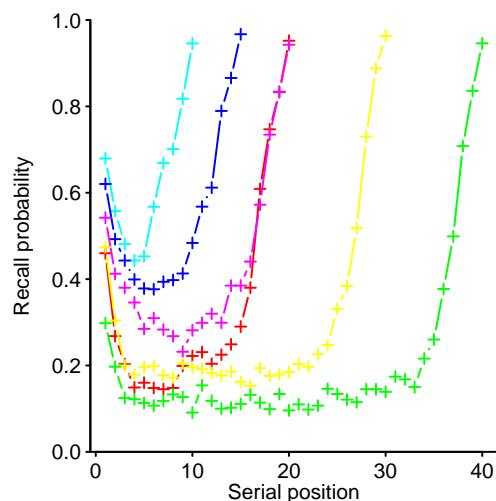


Figure 2.26: Probability of correct recall of words, by serial presentation order; for lists of length 10, 15 and 20 each word visible for 1, for lists of length 20, 30 and 40 each word visible for 2 seconds. Data from Murdoch,<sup>1119</sup> via Brown.<sup>220</sup> [code](#)

Subjects were free to recall the previously seen lines in any order. An analysis of the recall order showed that students grouped lines by syntactic construct (e.g., loop, function header), while the course teachers recalled the lines in the order in which they appeared in the three programs (i.e., they reconstructed three programs from the 16 randomly ordered lines); see Figure 2.27.

A study by Pennington<sup>1232</sup> found that developers responded slightly faster to questions about a source code statement when its immediately preceding statement made use of closely related variables.

## 2.4.5 Forgetting

People are unhappy when they forget things; however, not forgetting may be a source of unhappiness.<sup>1164</sup> The Russian mnemonist Shereshevskii found that his ability to remember everything cluttered up his mind.<sup>991</sup> Having many similar, not recently used, pieces of information matching during a memory search can be counterproductive; forgetting is a useful adaptation.<sup>1391</sup> For instance, a driver returning to a car wants to know where it was last parked, not the location of all previous parking locations. It has been proposed that human memory is optimized for information retrieval based on the statistical properties of the likely need for the information,<sup>44</sup> in peoples' everyday lives (such as the pattern of book borrowings in libraries<sup>233</sup>). The rate at which the mind forgets seems to mirror the way that information tends to lose its utility, over time, in the world in which we live. Organizational forgetting is discussed in Section 3.4.3.

Some studies of forgetting have found that a power law is a good fit to the reduction, over time, in subjects' ability to correctly recall information,<sup>1360</sup> while results from other studies are better fitted by an exponential equation (over the measurement range of many experiments, the difference between the two fitted models is usually small).<sup>v</sup> As more experimental data has become available, more complicated models have been proposed.<sup>1060</sup>

A study by Ebbinghaus,<sup>427</sup> using himself as a subject, performed what has become a classic experiment in forgetting. The measure of learning and forgetting used was based on the relative time saved, when relearning previously learned information, compared to the time taken to first learn the information; Ebbinghaus learned lists of 104 nonsense syllables, with intervals between relearning of 20 minutes, 1 hour, 9 hours, 1 day, 2 days, 6 days and 31 days. Figure 2.28 shows the fraction of time saved, after a given duration, when relearning list contents to the original learned standard (with standard errors)

Another measure of information retention was used in a study by Rubin, Hinton and Wenzel.<sup>1359</sup> Subjects saw a pair of words on a screen, which they had to remember; later, when the first word appeared on the screen, they had to type the second word of the pair. Subjects saw a sequence of different word pairs; *lag* is defined as the number of words seen between first seeing a word pair and being asked to give the second word in response to the appearance of the first word of the pair.

Figure 2.29 shows the fraction of correct second words at various lag intervals. The red line is a fitted bi-exponential regression model, with blue and green lines showing its two exponential components.

A study by Meeter, Murre and Janssen<sup>1060</sup> modeled the likelihood of prominent news stories being correctly recalled over a period of 16 months. The questions had two possible forms: Forced choice, where four possible answers were listed and one had to be selected, and Open, where an answer had to be provided with no suggestions listed. Data was collected from 14,000 people, via an internet news test. Figure 2.30 shows the fraction of correct answers each day, against time elapsed since the event in the question was reported.

## 2.5 Learning and experience

People have the ability to learn, and on many tasks human performance improves with practice, e.g., time to deciding whether a character sequence is an English word;<sup>837</sup> many studies have fitted a power law to measurements of practice performance (the term *power law of*

<sup>v</sup> It has been suggested that the power laws are a consequence of fitting data averaged over multiple subjects; see Section 9.3.3.

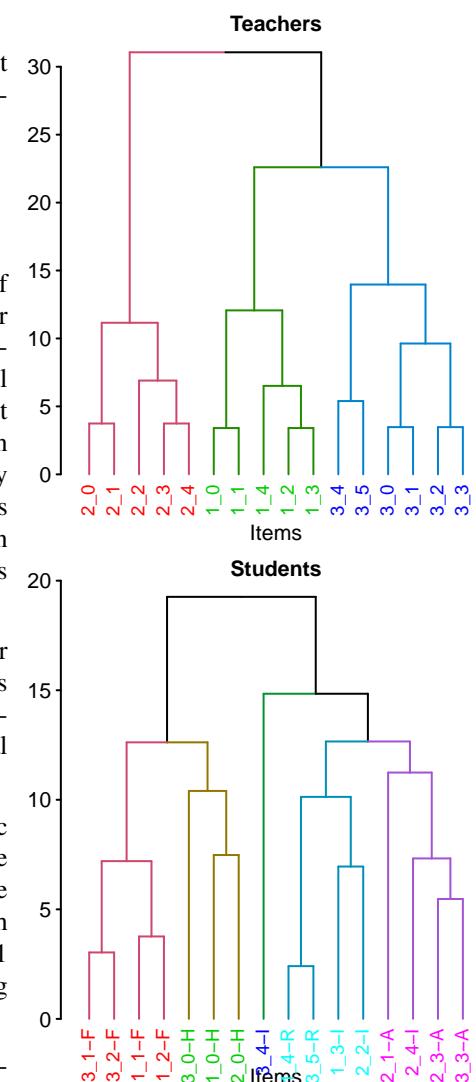


Figure 2.27: Hierarchical clustering of statement recall order, averaged over teachers and then students; label names are: program\_list-statementkind, where statementkind might be a function header, loop, etc. Data extracted from Adelson.<sup>8</sup> code

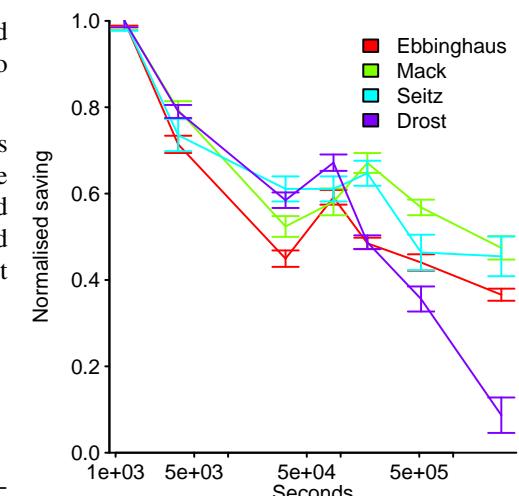


Figure 2.28: Fraction of relearning time saved (normalised) after given interval since original learning; original Ebbinghaus study and three replications (with standard errors). Data from Murre et al.<sup>1121</sup> code

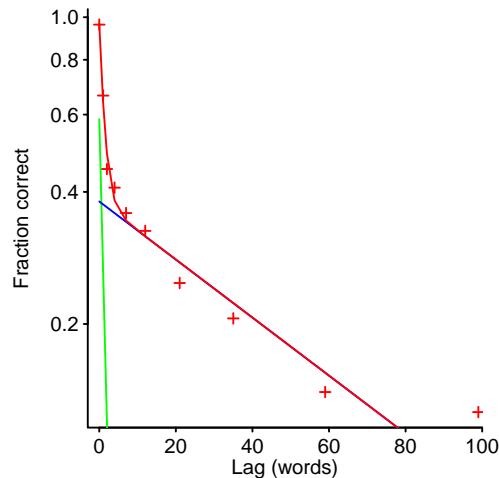


Figure 2.29: Fraction of correct subject responses, with fitted bi-exponential model in red (blue and green lines are its two exponential components). Data from Rubin et al.<sup>1359</sup> [code](#)

*learning* is often used). If chunking is assumed to play a part in learning, a power law is a natural consequence;<sup>1149</sup> the equation has the form:<sup>vi</sup>

$$RT = a + bN^{-c}$$

where:  $RT$  is the response time,  $N$  is the number of times the task has been performed, with  $a$ ,  $b$ , and  $c$  constants selected by model fitting.

There are also theoretical reasons for expecting the measurements to be fitted by an equation having an exponential form, and this form of model has been fitted to many learning data sets;<sup>673</sup> the equation has the form:<sup>vii</sup>

$$RT = a + be^{-cN}$$

Implicit learning occurs when people perform a task containing information that is not explicitly obvious to those performing it. A study by Reber and Kassin<sup>1317</sup> compared implicit and explicit pattern detection. Subjects were asked to memorize sets of words, with the words in some sets containing letter sequences generated using a finite state grammar. One group of subjects thought they were just taking part in a purely memory-based experiment, while the second group were told of the existence of a letter sequence pattern in some words, and that it would help their performance if they could deduce this pattern. The performance of the two groups, on the different sets of words (i.e., pattern words only, pattern plus non-pattern words, non-pattern words only) matched each other. Without being told to do so, subjects had used patterns in the words to help perform the memorization task.

Explicit learning occurs when the task contains patterns that are apparent, and can be remembered and used on subsequent performances. A study by Alteneder<sup>30</sup> recorded the time taken, by the author, to solve the same jig-saw puzzle 35 times (over a four-day period). After two weeks, the same puzzle was again solved 35 times. The exponent of the fitted power law, for the first series, is -0.5; Figure 2.31 show both a fitted power law and exponential.

Social learning, learning from others, is discussed in Section 3.4.2, and organizational learning is discussed in Section 3.4.3.

For simple problems, learning can result in the solution being committed to memory; performance is then driven by reaction-time, i.e., the time needed to recall and give the response. Logan<sup>976</sup> provides an analysis of subject performance on these kinds of problems.

The amount of practice needed to learn any patterns present in a task (to be able to perform it), depends on the complexity of these patterns. A study by Kruschke<sup>886</sup> asked subjects to learn the association between a stimulus and a category; they were told that the category was based on height and/or position. During the experiment subjects were told whether they had selected the correct category for the stimulus. When the category involved a single attribute (i.e., one of height or location), learning was predicted to be faster, compared to when it involved two attributes (i.e., learning difficulty depends on the number of variables and states).

Figure 2.32 shows the probability of a correct answer, for a particular kind of stimulus (only height or position, and some combination of height and position), for eight successive blocks of eight stimulus/answer responses.

The patterns present in a task may change. What impact do changes in task characteristics have on performance? A study by Kruschke<sup>887</sup> asked subjects to learn the association between a stimulus, and a category described by three characteristics (which might be visualized in a 3-D space; see Figure 2.44); once their performance was established at close to zero errors, the characteristics of the category were changed. The change pattern was drawn from four possibilities: reversing the value of each characteristic (considered to be a zero-dimension change), changes to one characteristic, two characteristics and three characteristics (not reversal).

Figure 2.33 shows average subject performance improving to near perfect (22 blocks successive blocks of eight stimulus/answer responses), a large performance drop when the category changes, followed by improving performance, over successive blocks, as subjects learn the new category. The change made to the pattern for the learned category has a noticeable impact on the rate at which the new category is learned.

<sup>vi</sup> Power laws can be a consequence of fitting data averaged over multiple subjects, rather than representing individual subject performance; see Section 9.3.3.

<sup>vii</sup> Similar equations can also be obtained by averaging over a group of individuals whose learning takes the form of a step function.<sup>539</sup>

The source code for an application does not have to be rewritten every time somebody wants a new copy of the program; it is rare for a developer to be asked to reimplement exactly the same application again. However, having the same developer reimplement the same application multiple times, provides information about the time savings that learning can provide.

A study by Lui and Chan<sup>985</sup> asked 24 developers to implement the same application four times; 16 developers worked in pairs (i.e., eight pair programming teams) and eight worked solo. Before starting to code, the subjects took a test involving 50 questions from a computer aptitude test; subjects were ranked by number of correct answers, and pairs selected such that both members were adjacent in the ranking.

Learning occurs every-time the application is implemented, and forgetting occurs during the period between implementations (each implementation occurred on a separate weekend, with subjects doing other work during the week). Each subject had existing knowledge and skill, which means everybody started the experiment at a different point on the learning curve. In the following analysis the test score is used as a proxy for each subject's initial point on the learning curve.

Figure 2.34 shows the completion times, for each round of implementation, for solo and pairs. The equation:  $Completion\_time = a \times (b \times Test\_score + Round)^c$ , fits to better than 0.05 statistical significance (where:  $Completion\_time$  is time to complete an implementation of the application,  $Test\_score$  the test score and  $Round$  the number of times the application has been implemented, with  $a$ ,  $b$  and  $c$  are constants chosen by the model fitting process). However, its predictions are in poor agreement with actual values, suggesting that other factors are making a significant contribution to performance.

After starting work in a new environment, performance can noticeable improve as a person gains experience working in that environment. A study by Brooks<sup>218</sup> measured the performance of an experienced developer, writing and debugging 23 short programs (mean length 24 lines). The time taken to debug each program improved as more programs were implemented (see [developers/a013582.R](#)).

Developers sometimes work in several programming languages on a regular basis. The extent to which learning applies across languages, is likely to be dependent on the ease with which patterns of usage are applicable across the languages used.

A study by Zislis<sup>1716</sup> measured the time taken (in minutes) by himself, to implement 12 algorithms, with the implementation performed three times using three different languages (APL, PL/1, Fortran), and on the fourth repetition using the same language as on the first implementation. Figure 2.35 shows total implementation time for each algorithm, over the four implementations; fitting a mixed-effects model finds some performance difference between the languages used (see figure code for details).

A study by Jones<sup>786</sup> investigated developer beliefs about binary operator precedence. Subjects were show an expression containing two binary operators, and had to specify the relative precedence of these operators by adding parenthesis to the expression, e.g.,  $a + b | c$ . In a coding environment, the more frequently a pair of binary operators appear together, the more often developers have to recall information about their precedence; the hypothesis was that the more often developers have to make a particular precedence decision, when reading code, the more likely they are to know the correct answer. Binary operator usage in code was used as a proxy for developer experience of making binary operator decisions (C source code was measured). Figure 2.36 shows the fraction of correct answers to the relative operator precedence question, against the corresponding percentage occurrence of that pair of binary operators.

A study by Mockus and Weiss<sup>1100</sup> found that the probability of developer introducing a fault into an application, when modifying the software, decreased as the log of the total number of changes made by the developer (i.e., their experience or expertise).

Job advertisements often specify that a minimum number of years of experience is required. Number of years may not be a reliable measure of expertise, but it does provide a degree of comfort that a person has had to deal with many of the problems that might occur within a given domain.

A study by Latorre<sup>925</sup> investigated the effect of developer experience on applying unit-test-driven development. The 24 subjects, classified as junior (one-to-two-years professional experience), intermediate (three-to-five-years experience) or senior (more than six-years experience), were taught unit-test-driven development, and the time taken for them to implement

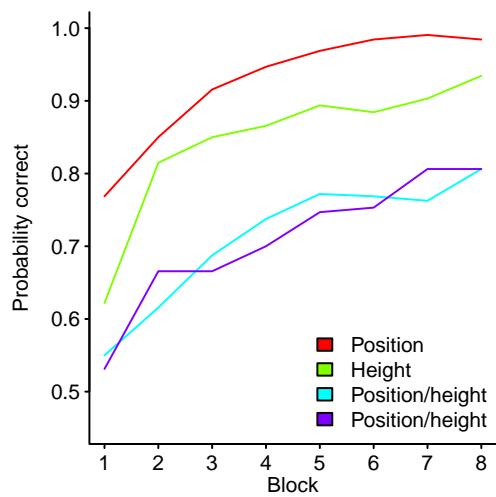


Figure 2.32: Probability of assigning a stimulus to the correct category, where the category involved: height, position, and a combination of both height and position. Data from Kruschke.<sup>886</sup> [code](#)

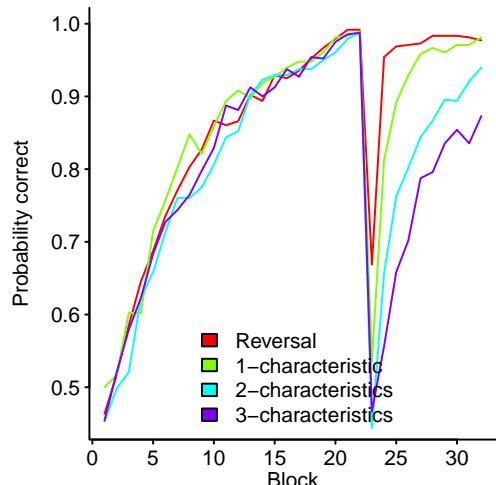


Figure 2.33: Probability of assigning a stimulus to the correct category; learning the category, followed in block 23 by a change in the characteristics of the learned category. Data from Kruschke.<sup>887</sup> [code](#)

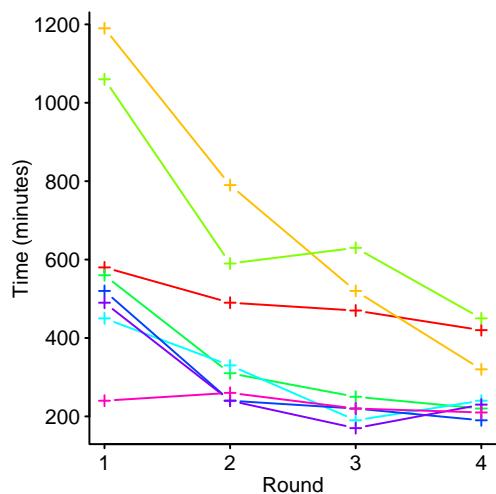


Figure 2.34: Completion times of eight solo developers for each implementation round. Data kindly provided by Lui.<sup>985</sup> [code](#)

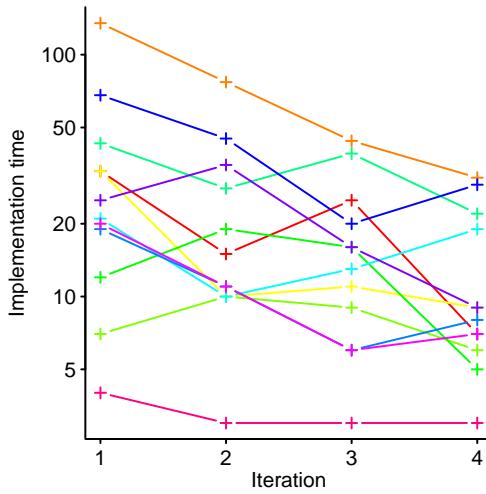


Figure 2.35: Time taken, by the same person, to implement 12 algorithms from the Communications of the ACM (each colored line), with four iteration of the implementation process. Data from Zislis.<sup>1716</sup> code

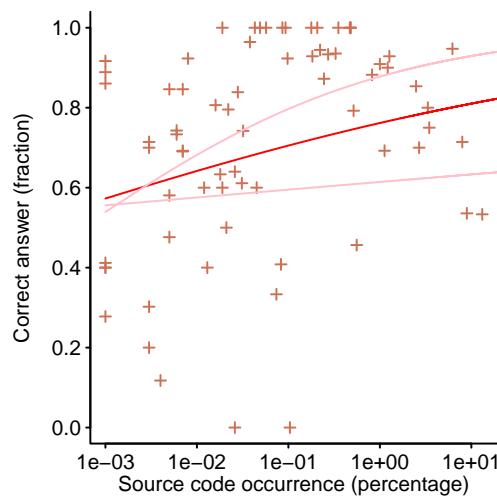


Figure 2.36: Percentage occurrence of binary operator pairs (as a percentage of all such pairs) against the fraction of correct answers to questions about their precedence, red line is beta regression model, plus bootstrapped 95% confidence intervals. Data from Jones.<sup>786</sup> code

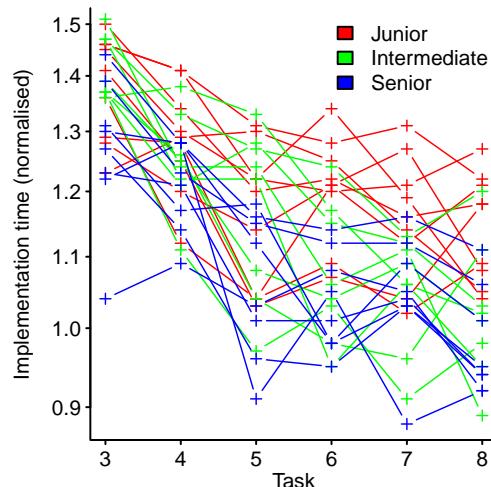


Figure 2.37: Time taken by 24 subjects, classified by years of professional experience, to complete successive tasks. Data from Latorre.<sup>925</sup> code

eight groups of requirements was measured. The implementation of the first two groups was included as part of the training and familiarization process; the time taken, by each subject, on these two groups was used to normalise the reported results.

Figure 2.37 shows the normalised time taken, by each subject, on successive groups of requirements; color is used to denote subject experience. While there is a lot of variation between subjects, average performance improves with years of experience, i.e., implementation time decreases (a fitted mixed-effects model is included in the code).

#### Clerical task<sup>1435</sup> ... emailed for data

What is the long term impact of learning on closely related cognitive activities? Isaac Asimov was a prolific author who maintained a consistent writing schedule. Figure 2.38 shows the number of published books against elapsed months. The lines are the fitted polynomials  $books \approx 27months^{0.48}$  (also a power-law), and  $books \approx -0.6months + 40\sqrt{months}$  (a better fit).

To quote Herbert Simon:<sup>1445</sup> “Intuition and judgement—at least good judgement—are simply analyses frozen into habit and into the capacity for rapid response through recognition. . . . Every manager needs also to be able to respond to situations rapidly, a skill that requires the cultivation of intuition and judgement over many years of experience and training.”

### 2.5.1 Belief

People hold beliefs derived from the information they have received and analysis of accumulated information over time.

#### How are existing beliefs modified by the introduction of new evidence?

A study by Hogarth and Einhorn<sup>713</sup> investigated order, and response mode effects in belief updating. Subjects were presented with a variety of scenarios (e.g., a defective stereo speaker thought to have a bad connection, or a baseball player whose hitting improved dramatically after a new coaching program). Subjects read an initial description followed by two or more additional items of evidence. The additional evidence was either positive (e.g., “The other players on Sandy’s team did not show an unusual increase in their batting average over the last five weeks”), or negative (e.g., “The games in which Sandy showed his improvement were played against the last-place team in the league”). The positive and negative evidence was worded to create either strong or weak forms.

The evidence was presented in three combinations: strong positive and then weak positive, upper plot in Figure 2.39; strong negative and then weak negative, middle plot of Figure 2.39; positive negative and then negative positive, lower plot of Figure 2.39. Subjects were asked, “Now, how likely do you think X caused Y on a scale of 0 to 100?” For some presentations, subjects had to respond after seeing each item of evidence (the step-by-step procedure), in the other presentations, subjects did not respond until seeing all the evidence (the end-of-sequence procedure).

Other studies have replicated these results, for instance, professional auditors have been shown to display recency effects in their evaluation of the veracity of company accounts.<sup>1218</sup>

Studies<sup>1351</sup> have found that people exhibit a belief preservation effect; they continue to hold beliefs after the original basis for those beliefs no longer holds.

The mathematical approach used to model quantum mechanics is started being used to model some cognitive processes, such as order effects and holding conflicting beliefs at the same time.<sup>1564</sup>

### 2.5.2 Expertise

The term *expert* might be applied to a person because of what other people think, i.e., be socially based, such as professional standing within an organization<sup>viii</sup>, and self-proclaimed experts willing to accept money from clients who are not willing to take responsibility for proposing what needs to be done<sup>55</sup> (e.g., the role of court jester who has permission to say what others cannot); or, be applied to a person because of what they can do, i.e., performance based, such as knowing a great deal about a particular subject, or being able to perform at

<sup>viii</sup> Industrial countries use professionalism as a way of institutionalising expertise.

a qualitatively higher level than the average person, or some chosen top percentage, or be applied to a person having a combination of social and performance skills.<sup>577</sup>

This section discusses expertise as a high-performance skill; something that requires many years of training, and where many individuals fail to develop proficiency.

How might people become performance experts?

Chess players were the subject of the first major study of expertise, by de Groot,<sup>366</sup> and techniques used to study Chess, along with the results obtained, continue to dominate the study of expertise. In a classic study, de Groot briefly showed subjects the position of an unknown game and asked them to reconstruct it. The accuracy and speed of experts (e.g., Grand Masters) was significantly greater than non-experts, when the pieces appeared on the board in positions corresponding to a game, but was not much greater when the pieces were placed at random. The explanation given for the significant performance difference, is that experts are faster to recognise relationships between the positions of pieces, and make use of their large knowledge of positional patterns to reduce the amount of working memory needed to remember what they were briefly shown.

A study by McKeithen, Reitman, Ruster and Hirtle<sup>1048</sup> gave subjects two minutes to study the source code of a program, and then gave them three minutes to recall the 31 lines of the program. Subjects were then given another two minutes to study the same source code, and asked to recall the code; this process was repeated for a total of five trials.

Figure 2.40 shows the number of lines recalled by experts (over 2,000 hours of general programming experience), intermediates (just completed a programming course), and beginners (about to start a programming course) over the five trials. The upper plot are the results for the 31 line program, and the lower plot a scrambled version of the program.

Some believe that experts have an innate ability, or capacity, that enables them to do what they do so well. Research has shown that while innate ability can be a factor in performance (there do appear to be genetic factors associated with some athletic performances), the main factor in developing expert performance is time spent in *deliberate practice*<sup>451</sup> (deliberate practice does not explain everything<sup>653</sup>).

Deliberate practice differs from simply performing the task, in that it requires people to monitor their practice with full concentration, and to receive feedback<sup>714</sup> on what they are doing (often from a professional teacher). The goal of deliberate practice being to improve performance, not to produce a finished product, and may involve studying components of the skill in isolation, attempting to improve on particular aspects. The goal of this practice being to improve performance, not to produce a finished product.

In many fields expertise is acquired by memorizing a huge amount of domain-specific information, organizing it for rapid retrieval based on patterns that occur when problem solving within the domain and refining the problem solving process.<sup>453</sup>

Studies of the backgrounds of recognized experts, in many fields, found that the elapsed time between them starting out, and carrying out their best work was at least 10 years, often with several hours of deliberate practice every day of the year. For instance, a study of violinists<sup>452</sup> (a perceptual-motor task), found that by age 20 those at the top-level had practiced for 10,000 hours, those at the next level down 7,500 hours, and those at the lowest level of expertise had practiced for 5,000 hours; similar quantities of practice were found in those attaining expert performance levels in purely mental activities (e.g., chess).

Expertise within one domain does not confer any additional skills within another domain,<sup>42</sup> e.g., statistics (unless the problem explicitly involves statistical thinking within the applicable domain), and logic.<sup>283</sup> A study<sup>277</sup> in which subjects learned to remember long sequences of digits (after 50–100 hours of practice they could commit to memory, and later recall, sequences containing more than 20 digits) found that this expertise did not transfer to learning sequences of other items.

There are domains in which those acknowledged as experts do not perform significantly better than those considered to be non-experts,<sup>243</sup> in some cases non-experts have been found to outperform experts within their domain.<sup>1665</sup> An expert's domain knowledge can act as a mental set that limits the search for a solution; the expert becomes fixated within the domain; in cases where a new task does not fit the pattern of highly proceduralized behaviors of an expert, a novice has an opportunity to do better.

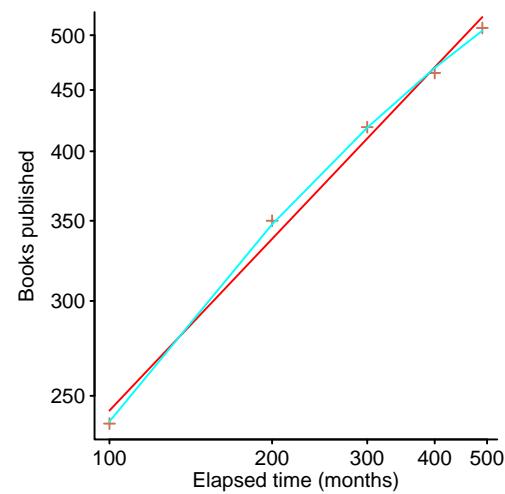


Figure 2.38: Elapsed months during which Asimov published a given number of books, with lines for two fitted regression models. Data from Ohlsson.<sup>1183</sup> [code](#)

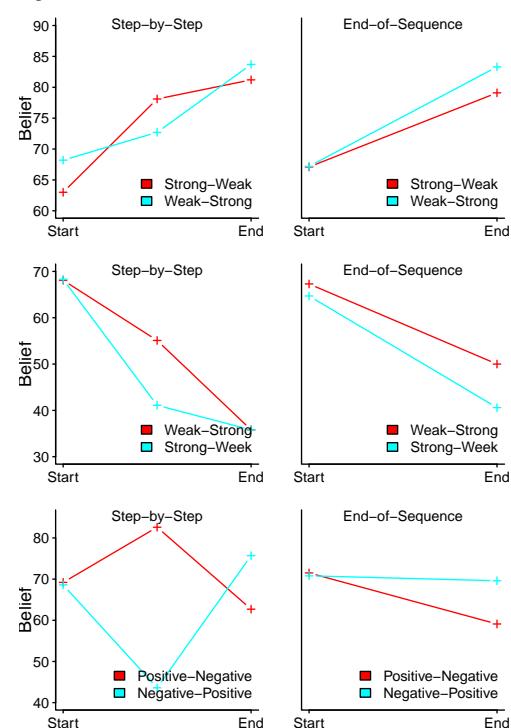


Figure 2.39: Subjects' belief response curves when presented with evidence in the sequences: (upper) positive weak, then positive strong, (middle) negative weak then negative strong, (lower) positive then negative. Based on Hogarth et al.<sup>713</sup> [code](#)

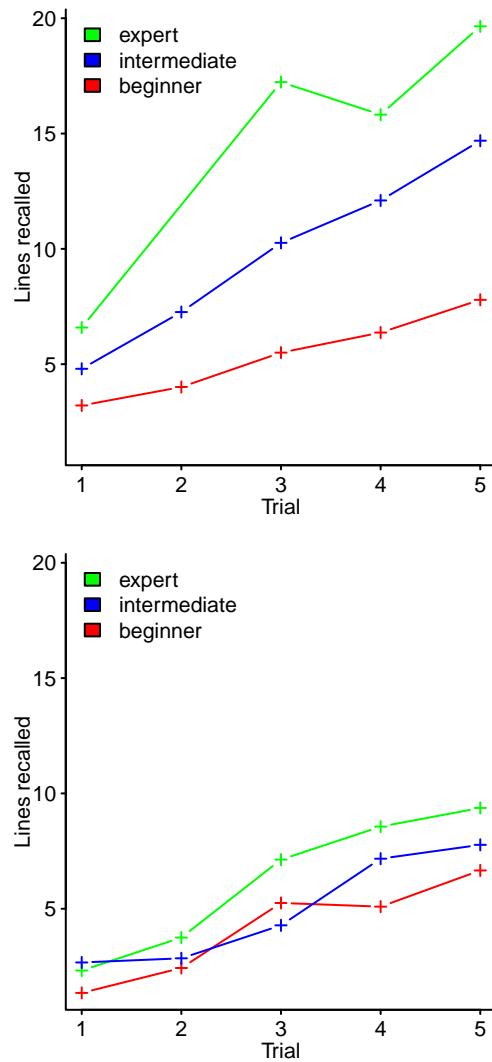


Figure 2.40: Lines of code correctly recalled after a given number of 2 minute memorization sessions; upper plot actual program, lower plot line order scrambled. Data extracted from McKeithen et al.<sup>1048</sup> [code](#)

What of individual aptitudes? In the cases studied, the effects of aptitude, if there were any, were found to be completely overshadowed by differences in experience, and deliberate practice times. Willingness to spend many hours, every day, studying to achieve expert performance is certainly a necessary requirement. Does an initial aptitude, or interest, in a subject lead to praise from others (the path to musical and chess expert performance often starts in childhood), which creates the atmosphere for learning, or are other issues involved? IQ scores do correlate to performance during, and immediately after training, but the correlation reduces over the years. The IQ scores of experts has been found to be higher than the average population, at about the level of college students.

Education can be thought of as trying to do two things (of interest to us here)—teach students skills (procedural knowledge), and providing them with information, considered important in the relevant field, to memorize (declarative knowledge).

Does attending courses in particular subjects have any measurable effect on students' capabilities, other than being able to answer questions in an exam? That is, having developed some skill in using a particular system of reasoning, do students apply it outside of the domain in which they learnt it?

A study by Lehman, Lempert, and Nisbett<sup>941</sup> measured changes in students' statistical, methodological and conditional reasoning abilities (about everyday-life events) between their first and third years. They found that both psychology and medical training produced large effects on statistical and methodological reasoning, while psychology, medical and law training produced effects on the ability to perform conditional reasoning; training in chemistry had no effect on the types of reasoning studied. An examination of the skills taught to students studying in these fields showed that they correlated with improvements in the specific types of reasoning abilities measured.

Fluency in use of tools of the trade...

A study by Remington, Yuen and Pashler<sup>1325</sup> compared subject performance between using a GUI and a command line (with practice, there was little improvement in GUI performance, but command line performance continued to improve and eventually overtook GUI performance). Figure 2.41 shows the command line response time for one subject over successive blocks of trials, and a fitted loess line.

?

### 2.5.3 Category knowledge

Children as young as four have been found to use categorization to direct the inferences they make,<sup>556</sup> and many studies have shown that people have an innate desire to create and use categories (they have also been found to be sensitive to the costs and benefits of using categories;<sup>1006</sup> Capuchin monkeys have learned to classify nine items concurrently<sup>1046</sup>). By dividing items into categories, people reduce the amount of information they need to learn,<sup>1271</sup> and can generalize based on prior experience.<sup>1161</sup> Information about the likely characteristics of a newly encountered item can be obtained by matching it to one or more known categories, and then extracting characteristics common to previously encountered items in these categories. For instance, a flying object with feathers, and a beak might be assigned to the category *bird*, which suggests the characteristics of laying eggs and being migratory.

Categorization is used to perform inductive reasoning (the derivation of generalized knowledge from specific instances), and also acts as a memory aid (remembering the members of a category). Categories provide a framework from which small amounts of information can be used to infer, seemingly unconnected (to an outsider), useful conclusions.

Studies have found that people use roughly three levels of abstraction in creating hierarchical relationships. The highest level of abstraction has been called<sup>1346</sup> the *superordinate-level* (e.g., the general category furniture), the next level down the *basic-level* (this is the level at which most categorization is carried out, e.g., car, truck, chair or table), the lowest level is the *subordinate-level* (which denotes specific types of objects, e.g., a family car, a removal truck, my favourite armchair, a kitchen table). Studies<sup>1346</sup> have found that basic-level categories have properties not shared by the other two categories; adults spontaneously name objects at this level, it is the abstract level that children acquire first, and category members tend to have similar overall shapes.

When categories have a hierarchical structure, it is possible for an attribute of a higher-level category to affect the perceived attributes of subordinate categories. This is illustrated in a study by Stevens and Coupe,<sup>1506</sup> in which subjects were asked to remember the information contained in a series of maps (see Figure 2.42). They were asked questions such as: “Is X east or west of Y?” and “Is X north or south of Y?” Subjects gave incorrect answers 18% of the time for the congruent maps, but 45% of the time for the incongruent maps (15% for homogeneous). These results were interpreted as information about the relative locations of countries influencing the answer questions about the city locations.

Studies<sup>1458</sup> have found that people do not consistently treat subordinate categories as inheriting the properties of their superordinates, i.e., category inheritance need not be a tree.

How categories should be defined and structured is a long-standing debate within the sciences. Some commonly used category formation techniques, their membership rules and attributes include:

- defining-attribute theory: members of a category are characterized by a set of defining attributes. Attributes divide objects up into different concepts whose boundaries are well-defined, with all members of the concept being equally representative. Concepts that are a basic-level of a superordinate-level concept, will have all the attributes of that superordinate level; for instance, a sparrow (small, brown) and its superordinate bird (two legs, feathered, lays eggs),
- prototype theory: categories have a central description, the *prototype*, that represents the set of attributes of the category. This set of attributes need not be necessary, or sufficient, to determine category membership. The members of a category can be arranged in a typicality gradient, representing the degree to which they represent a typical member of that category. It is possible for objects to be members of more than one category (e.g., tomatoes as a fruit, or a vegetable),
- exemplar-based theory: specific instances, or *exemplars*, act as the prototypes against which other members are compared. Objects are grouped, relative to one another, based on some similarity metric. The exemplar-based theory differs from the prototype theory in that specific instances are the norm against which membership is decided. When asked to name particular members of a category, the attributes of the exemplars are used as cues to retrieve other objects having similar attributes,
- explanation-based theory: there is an explanation for why categories have the members they do. For instance, the biblical classification of food into *clean* and *unclean* is roughly explained the correlation between type of habitat, biological structure, and form of locomotion; creatures of the sea should have fins, scales and swim (sharks and eels don’t), and creatures of the land should have four legs (ostriches don’t).

From a predictive point of view, explanation-based categories suffer from the problem that they may heavily depend on the knowledge and beliefs of the person who formed the category; for instance, the set of objects a person would remove from their home, if it suddenly caught fire.

Figure 2.43 shows the possible combinations of three, two-valued attributes, color/size/shape; there are eight possibilities. It is possible to create six unique categories by selecting four items from these eight possibilities (see Figure 2.44; there are 70 different ways of taking four things from a choice of eight,  $8!/(4!4!)$ ), and taking symmetry into account reduces the number of unique categories to six).

A study by Shepard, Hovland, and Jenkins<sup>1425</sup> measured subject performance in assigning objects to these six categories. Subject error rate decreased with practice.

Estes<sup>459</sup> proposed the following method for calculating the similarity of two objects. Matching attributes have the similarity coefficient,  $0 \leq t \leq \infty$ , and nonmatching attributes have similarity coefficient,  $0 \leq s_i \leq 1$  (potentially different for each nonmatch), is assigned for each nonmatching attribute. When comparing objects within the same category, the convention is to use  $t = 1$ , and to give the attributes that differ the same similarity coefficient,  $s$ .

As an example, consider just two of the attributes shape/color in Figure 2.43, giving the four combinations red/green—triangles/squares; assign red-triangle and red-square to category A, assign green-triangle and green-square to category B, i.e., category membership is decided by color. The similarity of each of the four possible object combinations to category A and

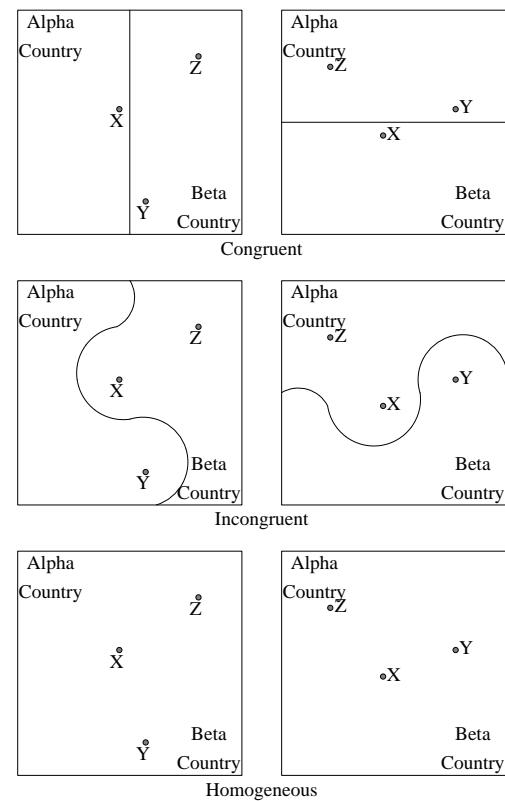


Figure 2.42: Country boundaries distort judgement of relative city locations. Based on Stevens et al.<sup>1506</sup>

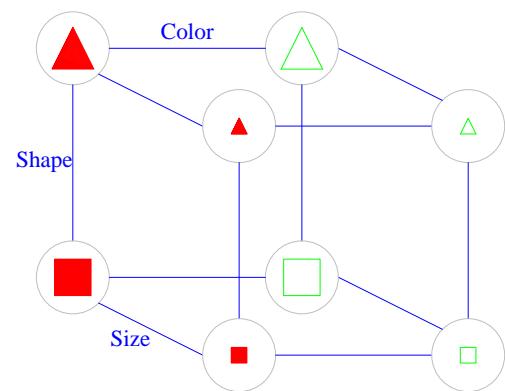


Figure 2.43: Orthogonal representation of shape, color and size stimuli. Based on Shepard.<sup>1425</sup>

B is listed in Table 2.3. Looking at the top row: red-triangle is compared for similarity to all members of category A ( $1 + s$ , because it does not differ from itself and differs in one attribute from the other member of category A), and all members of category B ( $s + s^2$ , because it differs in one attribute from one member of category B and in two attributes from the other member).

Stimulus	Similarity to A	Similarity to B
Red triangle	$1 + s$	$s + s^2$
Red square	$1 + s$	$s + s^2$
Green triangle	$s + s^2$	$1 + s$
Green square	$s + s^2$	$1 + s$

Table 2.3: Similarity of a stimulus object to: category A: red triangle and red square; category B: green triangle and green square.

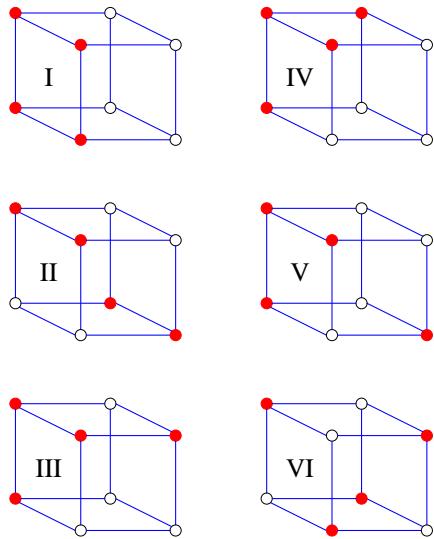


Figure 2.44: The six unique configurations of selecting four times from eight possibilities, i.e., it is not possible to rotate one configuration into another within these six configurations. Based on Shepard.<sup>1425</sup>

If a subject is shown a stimulus that belongs in category A, the expected probability of them assigning it to this category is:  $\frac{1+s}{(1+s)+(s+s^2)} \rightarrow \frac{1}{1+s}$ . When  $s$  is 1, the expected probability is no better than a random choice; when  $s$  is 0, the probability is a certainty.

A study by Feldman<sup>483</sup> involved categories containing objects having either three or four attributes. During an initial training period subjects learned to distinguish between a creature having a given set of attributes, and other creatures that did not. Subjects then saw a sequence of example creatures, and had to decide whether they were a member of the learned category.

A later study<sup>484</sup> specified category membership algebraically, e.g., membership of category IV in the top right of Figure 2.44 is specified by the expression:  $\overline{SHC} + SH\overline{C} + S\overline{HC} + SHC$ , where:  $S$  is size,  $H$  is shape,  $C$  is color, and an *overline* indicates negation. The number of terms in the minimal boolean formula specifying the category (a measure of category complexity, which Feldman terms *boolean complexity*) was found to predict the trend in subject error rate (i.e., number of errors in selecting the correct category increased with boolean complexity, see Figure 2.45).

There are a few human generated semantic classification datasets publically available.<sup>365</sup>

## 2.5.4 Categorization consistency

Obtaining benefits from using categories requires some degree of consistency in assigning items to categories. In the case of an individual's internal categories, the ability to consistently assign items to the appropriate category is required; within cultural groups there has to be some level of agreement between members over the characteristics of items in each category.

Cross-language research has found that there are very few concepts that might be claimed to be universal (they mostly relate to the human condition).<sup>1619, 1662</sup>

Culture, and the use of items, can play an important role in the creation and use of categories.

A study by Bailenson, Shum, Atran, Medin and Coley<sup>90</sup> compared the categories created for two sets (US and Maya) of 104 bird species, by three groups; subjects were US bird experts (average of 22.4 years bird watching), US undergraduates, and ordinary Itzaj (Maya Amerindians people from Guatemala). The categorization choices made by the three groups of subjects were found to be internally consistent within each group. The US experts' categories correlated highly with the scientific taxonomy for both sets of birds, the Itzaj categories only correlated highly for Maya birds (they used ecological justifications; the bird's relationship with its environment), and the nonexperts had a low correlation for both set of birds. Cultural differences were found in that, for instance, US subjects were more likely to generalise from songbirds, while the Itzaj were more likely to generalize from perceptually striking birds.

A study by Labov<sup>902</sup> showed subjects pictures of items that could be classified as either cups or bowls (see upper plot in Figure 2.46). These items were presented in one of two contexts—a neutral context in which the pictures were simply presented, and a food context (they were asked to think of the items as being filled with mashed potatoes).

The lower plot of Figure 2.46 shows that as the width of the item seen was increased, an increasing number of subjects classified it as a bowl. Introducing a food context, shifted subjects' responses towards classifying the item as a bowl, at narrower widths.

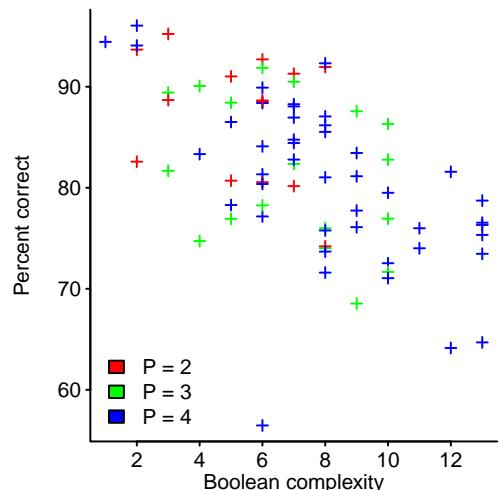


Figure 2.45: Percentage of correct answers given by one subject, against boolean-complexity of category, colored by number of positive cases needed to define the category. Data kindly provided by Feldman.<sup>483</sup> [code](#)

The same set of items may be viewed from a variety of different points of view (the term *frame* is sometimes used); for instance, commercial events include buying, selling, paying, charging, pricing, costing, spending, and so on. Figure 2.47 shows four ways (i.e., buying, selling, paying, and charging) of classifying the same commercial event.

A study by Jones<sup>789</sup> investigated the extent to which different developers make similar decisions when creating data structures to represent the same information; see Figure 12.4.

## 2.6 Reasoning

Reasoning enhances cognition. The knowledge-based use case for the benefits of being able to reason, is the ability it provides to extract information from available data; adding constraints on the data (e.g., known behaviors) can increase the quantity of information that may be extracted. Dual process theories<sup>1457, 1488, 1489</sup> treat people as having two systems: unconscious reasoning and conscious reasoning. What survival advantages does an ability to consciously reason provide, or is it primarily an activity WEIRD people need to learn to pass school tests?

Outside of classroom problems, a real world context in which people explicitly reason is decision-making, and here “fast and frugal algorithms”<sup>567, 570</sup> provide answers quickly, within the often limited constraints of time and energy. Context and semantics are crucial inputs to the reasoning process.<sup>1500</sup>

Understanding spoken language requires reasoning about what is being discussed,<sup>1066</sup> and in a friendly shared environment it is possible to fill in the gaps by assuming that what is said is relevant,<sup>1476</sup> with no intent to trick. In an adversarial context sceptical reasoning, of the mathematical logic kind, is useful for enumerating possible interpretations of what has been said.<sup>1067</sup>

The kinds of questions asked in studies of reasoning appear to be uncontentious. However, studies<sup>990, 1401</sup> of reasoning using illiterate subjects, from remote parts of the world, received answers to verbal reasoning problems that were based on personal experience and social norms, rather than the western ideal of logic. The answers given by subjects, in the same location, who had received several years of schooling were much more likely to match those demanded by mathematical logic; the subjects had learned how to act WEIRD and *play the game*. The difficulties experienced by those learning formal logic suggests that there is no innate capacity for this task (innate capacity enables the corresponding skill to be learned quickly and easily). The human mind is a story processor, not a logic processor.<sup>647</sup>

Reasoning and decision-making appear to be closely related. However, reasoning researchers tied themselves to using the norm of mathematical logic for many decades, and created something of research ghetto,<sup>1502</sup> while decision-making researchers have been involved in explaining real-world problems.

The Wason selection task<sup>1640</sup> is to studies of reasoning like the fruit fly is to studies of genetics. Wason’s study was first published in 1968, and considered mathematical logic to be the norm against which human reasoning performance should be judged. The reader might like to try this selection task:

- Figure 2.48 depicts a set of four cards, of which you can see only the exposed face but not the hidden back. On each card, there is a number on one of its sides, and a letter on the other.
- Below there is a rule, which applies only to the four cards. Your task is to decide, which, if any, of these four cards you must turn in order to decide if the rule is true.
- Don’t turn unnecessary cards. Tick the cards you want to turn.

**Rule:** If there is a vowel on one side, then there is an even number on the other side.

**Answer: ?** The failure of many subjects to give the expected answer (i.e., the one derived using mathematical logic) surprised many researchers, and over the years a wide variety of explanations, experiments, thesis, and books have attempted to explain the answers given. Explanations for subject behavior include: human reasoning is tuned for detecting people who cheat<sup>333</sup> within a group where mutual altruism is the norm, interpreting the wording

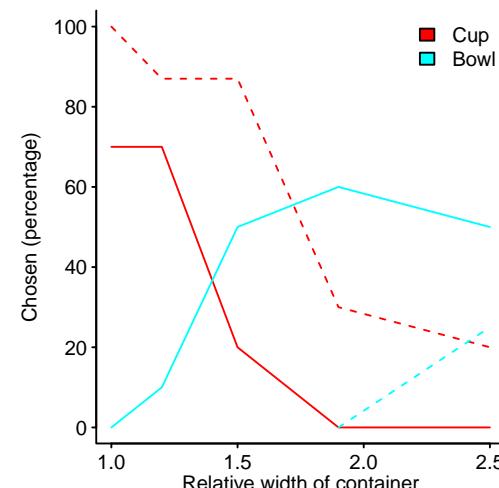
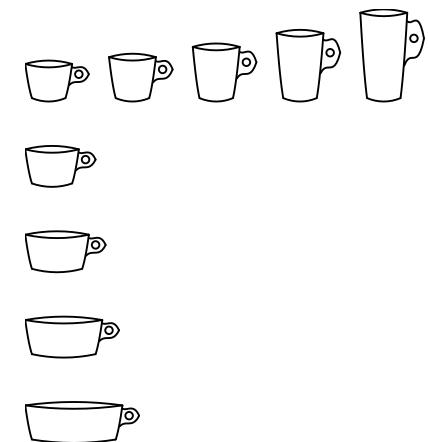


Figure 2.46: Cup- and bowl-like objects of various widths (ratios 1.2, 1.5, 1.9, and 2.5), and heights (ratios 1.2, 1.5, 1.9, and 2.4). The percentage of subjects who selected the term *cup* or *bowl* to describe the object they were shown (the paper did not explain why the figures do not sum to 100%). Based on Labov.<sup>902</sup> code

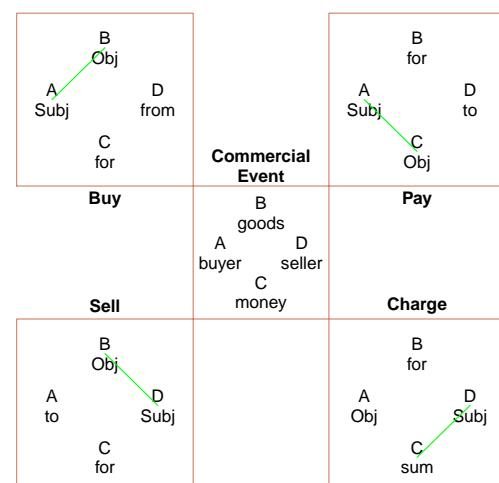


Figure 2.47: A commercial event involving a buyer, seller, money and goods; as seen from the buy, sell, pay, or charge perspective. Based on Fillmore.<sup>495</sup> code

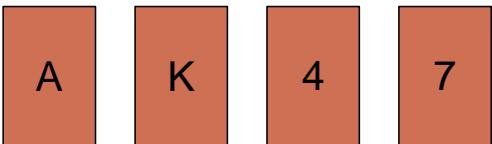


Figure 2.48: The four cards used in the Wason selection task. Based on Wason.<sup>1640</sup> code

Alfred North Whitehead: “It is a profoundly erroneous truism ellipsis; that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them.”

of questions pragmatically based on how natural language is used rather than as logical formula<sup>702</sup> (i.e., assuming a social context; people are pragmatic virtuosos rather than logical defectives), and adjusting norms to take into account cognitive resource limitations (i.e., computational and working memory), or a rational analysis approach.<sup>1175</sup>

Some Wason task related studies used a dialogue protocol (i.e., subjects’ discuss their thoughts about the problem with the experimenter), and transcriptions<sup>1501</sup> of these studies read like people new to programming having trouble understanding what it is they have to do to solve a problem by writing code.

People have different aptitudes, and this can result in them using different strategies to solve the same problem,<sup>1488</sup> e.g., an interaction between a subject’s verbal and spatial ability, and the strategy used to solve linear reasoning problems.<sup>1504</sup> However, a person having high spatial ability, for instance, does not necessarily use a spatial strategy. A study by Roberts, Gilmore and Wood<sup>1337</sup> asked subjects to solve what appeared to be a spatial problem (requiring the use of a very inefficient spatial strategy to solve). Subjects with high spatial ability used non-spatial strategies, while those with low spatial ability used a spatial strategy. The conclusion drawn was that those with high spatial ability were able to see the inefficiency of the spatial strategy, and selected an alternative strategy, while those with less spatial ability were unable to make this evaluation.

A study by Bell and Johnson-Laird<sup>136</sup> investigated the effect of the kind of questions asked on reasoning performance. Subjects had to give yes/no responses to two kinds of questions, asking about what is possible or what is necessary. The hypothesis was that subjects would find it easier to infer a *yes* answer to a question about what is possible, compared to one about what is necessary, because only one instance needs to be found (all instances need to be checked, to answer *yes* to a question about necessity). For instance, in a game in which only two can play, and the following information:

If Allan is in then Betsy is in.  
If Carla is in then David is out.

answering *yes* to the question “Can Betsy be in the game?” (a possibility), is easier than giving the same answer to “Must Betsy be in the game?” (a necessity); see Table 2.4.

Question	Correct <i>yes</i>	Correct <i>no</i>
is possible	91%	65%
is necessary	71%	81%

Table 2.4: Percentage of correct answers to the two kinds of questions and two kinds of response. Data from Bell et al.<sup>136</sup>

However, subjects would find it easier to infer a *no* answer to a question about what is necessary, compared to one about what is possible, because only one instance needs to be found, whereas all instances need to be checked to answer *no* to a question about possibility. For instance, in another two person game and the following information:

If Allan is out then Betsy is out.  
If Carla is out then David is in.

answering *no* to the question “Must Betsy be in the game?” (a necessity), is easier than giving the same answer to “Can Betsy be in the game?” (a possibility).

**Conditionals in English** In human languages the conditional clause generally precedes the conclusion, in a conditional statement.<sup>621</sup> An example where the conditional follows the conclusion is “I will leave, if you pay me” given as the answer to the question “Under what circumstances will you leave?”. In one study of English,<sup>519</sup> the conditional preceded the conclusion in 77% of written material, and 82% of spoken material. There is a lot of variation in the form of the conditional.<sup>157,253</sup>

## 2.6.1 Deductive reasoning

Deductive reasoning is the process of reasoning about one or more statements (technically known as *premises*) to reach one or more logical conclusions.

Studies<sup>85</sup> have found that the reasoning strategies used by people involve building either a verbal or spatial model, from the premises. Factors that have been found to affect peoples performance in solving deductive reasoning problems include the following:

- Belief bias: people are more willing to accept a conclusion, derived from given premises, that they believe to be true than one they believe to be false. A study by Evans, Barston, and Pollard<sup>460</sup> gave subjects two premises and a conclusion, and asked them to state whether the conclusion was true or false (based on the premises given; the conclusions were rated as either believable or unbelievable by a separate group of subjects); see Table 2.5,

Status-context	Example	Accepted
Valid-believable	No Police dogs are vicious Some highly trained dogs are vicious Therefore, some highly trained dogs are not police dogs	88%
Valid-unbelievable	No nutritional things are inexpensive Some vitamin tablets are inexpensive Therefore, some vitamin tablets are not nutritional things	56%
Invalid-believable	No addictive things are inexpensive Some cigarettes are inexpensive Therefore, some addictive things are not cigarettes	72%
Invalid-unbelievable	No millionaires are hard workers Some rich people are hard workers Therefore, some millionaires are not rich people	13%

Table 2.5: Percentage of subjects accepting that the stated conclusion could be logically deduced from the given premises. Based on Evans et al.<sup>460</sup>

- Form of premise: a study by Dickstein<sup>402</sup> measured subjects performance on the 64 possible two premise syllogisms (a premise being one of the propositions: *All S are P*, *No S are P*, *Some S are P*, and *Some S are not P*). For instance, the following syllogisms show the four possible permutations of three terms (the use of S and P is interchangeable):

$$\begin{array}{llll} \text{All M are S} & \text{All S are M} & \text{All M are S} & \text{All S are M} \\ \text{No P are M} & \text{No P are M} & \text{No M are P} & \text{No M are P} \end{array}$$

The results found that performance was affected by the order in which the terms occurred in the two premises of the syllogism. The order in which the premises are processed may affect the amount of working memory needed to reason about the syllogism, which in turn can affect human performance.<sup>573</sup>

## 2.6.2 Linear reasoning

Being able to make relational decisions is a useful skill for animals living in hierarchical social groups, where aggression is used to decide status.<sup>1223</sup> Aggression is best avoided, as it can lead to death or injury; the ability to make use of relative dominance information (obtained by watching interactions between other members of the group) may remove the need for aggressive behavior during an encounter between two group members who have not recently contested dominance (i.e., there is nothing to be gained in aggression towards a group member who has previously been seen to dominate a member who is dominant to yourself).

Another benefit of being ability to make relational comparisons, is being able to select, which of two areas contains the largest amount of food. Some animals, including humans, have a biologically determined representation of numbers, including elementary arithmetic operations, what one researcher has called the *number sense*.<sup>385</sup>

The use of relational operators have an obvious interpretation in terms of linear syllogisms. A study by De Soto, London, and Handel<sup>371</sup> investigated a task they called *social reasoning*, using the relations *better* and *worse*. Subjects were shown two relationship statements involving three people, and a possible conclusion (e.g., “Is Mantle worse than Moskowitz?”), and had 10 seconds to answer “yes”, “no”, or “don’t know”. The British National Corpus<sup>937</sup> lists *better* as appearing 143 times per million words, while *worse* appears under 10 times per million words, and is not listed in the top 124,000 most used words.

There are two patterns of performance behavior seen in the results; see Table 2.6:

<b>Relationships</b>	<b>Correct %</b>	<b>Relationships</b>	<b>Correct %</b>
1. A is better than B B is better than ++C++	60.5	5. A is better than B .5000 pluspluslog oS_xspace is worse than B++	61.8
2. B is better than ++C++		6. .5000 pluspluslog oS_xspace is worse than B++	
A is better than B	52.8	A is better than B	57.0
3. B is worse than A .5000 pluspluslog oS_xspace is worse than B++	50.0	7. B is worse than A B is better than ++C++	41.5
4. .5000 pluspluslog oS_xspace is worse than B++ B is worse than A	42.5	8. B is better than ++C++	
		B is worse than A	38.3

Table 2.6: Eight sets of premises describing the same relative ordering between A, B, and ++C++ (peoples names were used in the study) in different ways, followed by the percentage of subjects giving the correct answer. Based on De Soto et al.<sup>371</sup>

- a higher percentage of correct answers were given when the direction was better-to-worse (case 1), than mixed direction (case 2, 3), and were least correct in the direction worse-to-better (case 4),
- a higher percentage of correct answers were given when the premises stated an end term (better or worse) followed by the middle term, than a middle term followed by an end term.

A second experiment, in the same study, gave subjects printed statements about people. For instance, “Tom is better than Bill”. The relations used were *better*, *worse*, *has lighter hair*, and *has darker hair*. The subjects had to write the peoples names in two of four possible boxes; two arranged horizontally and two arranged vertically.

The results showed 84% of subjects selecting a vertical direction for better/worse, with better at the top (which is consistent with the *up is good* usage found in English metaphors<sup>906</sup>). In the case of lighter/darker 66% of subjects used a horizontal direction, with no significant preference for left-to-right or right-to left.

A third experiment in the same study used the relations *to-the-left* and *to-the-right*, and *above* and *below*. The above/below results were very similar to those for better/worse. The left-right results found that subjects learned a left-to-right ordering better than a right-to-left ordering.

The results of this study show the effect that operand order has on the accuracy of peoples responses. However, the interpretation placed on the operator also plays a significant role. Without knowing what interpretation a reader is likely to give to the operands and operators in the following two (logically equivalent) conditional expressions, for instance, it is not possible to select the one that is most likely to minimize incorrect reasoning on the part of readers.

```
if ((x <= y) && (x => z))
if ((x >= z) && (x <= y))
```

ACCU relational if-condition study...

Subject performance on linear reasoning improves, the greater the distance between the items being compared; the *distance effect* is discussed in Section 2.7.

### 2.6.3 Causal reasoning

A question often asked by developers, while reading source, is “what causes this /situation/event to occur?” Causal questions, such as this, are also common in everyday life. However, there has been relatively little mathematical research on causality (statistics deals with correlation; Pearl<sup>1225</sup> covers mathematical aspects), and little psychological research on causal reasoning.<sup>1460</sup>

It is sometimes possible to express a problem in either a causal or conditional form. A study by Sloman, and Lagnado<sup>1461</sup> gave subjects one of the following two reasoning problems, and associated questions:

- Causal argument form:

A causes B  
A causes C  
B causes D  
C causes D  
D definitely occurred

with the questions: “If B had not occurred, would D still have occurred?”, or “If B had not occurred, would A have occurred?”

- Conditional argument form:

If A then B  
If A then C  
If B then D  
If C then D  
D is true

with the questions: “If B were false, would D still be true?”, or “If B were false, would A be true?”.

Table 2.7 shows that subject performance depended on the form in which the problem was expressed.

Question	Causal	Conditional
D holds?	80%	57%
A holds?	79%	36%

Table 2.7: Percentage “yes” responses to various forms of questions (based on 238 responses). Based on Sloman et al.<sup>1461</sup>

Causal learning<sup>198</sup> ...

## 2.7 Number processing

Having a sense of quantity, and being able to judge the relative size of two quantities, provides a number of survival benefits, including being able to perform an approximate number of repetitions of some task (e.g., pressing a bar, see Figure 2.4), and deciding which of two clusters of food is the largest.

Being able to quickly enumerate small quantities is sufficiently useful for the brain to support preattentive processing of up to four, or so, items.<sup>1561</sup> When asked to enumerate how many dots are visible in a well-defined area, subjects’ response time depends on the number of dots; with between one and four dots, performance varies between 40 ms to 100 ms per dot, but with five or more dots, performance varies between 250 ms to 350 ms per dot. The faster process is known as *subitizing* (people effortlessly **see** the number of dots), while the slower process is called *counting*.

Studies have found that a variety of animals make use of an approximate mental number system (sometimes known as the *number line*; see Figure 2.4); the extent to which brains have a built-in number line, or existing neurons are repurposed through learning, is an active area of research.<sup>383, 1172</sup> Humans are the only creatures known to have a second system, one that can be used to represent numbers exactly: language.

A study by van Oeffelen and Vos<sup>1593</sup> investigated subjects’ ability to estimate the number of dots in a briefly displayed image (100 ms, i.e., not enough time to be able to count the dots). Subjects were given a target number, and had to answer yes/no whether they thought the image they saw contained the target number of dots. Figure 2.50 shows the probability of a correct answer for various target numbers, and a given difference between target number and number of dots displayed.

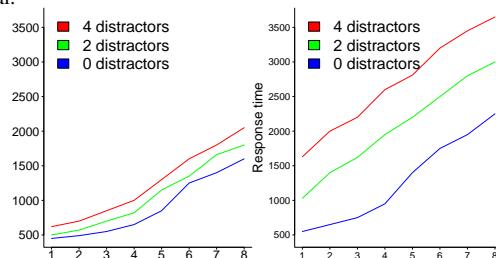
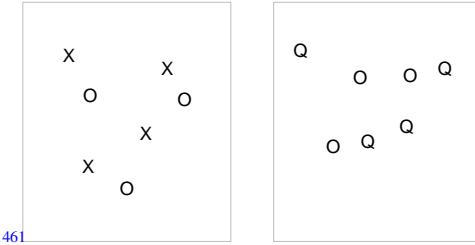


Figure 2.49: Average time (in milliseconds) taken for subjects to enumerate O’s in a background of X or Q distractors. Based on Trick and Pylyshyn.<sup>1561</sup> [code](#)

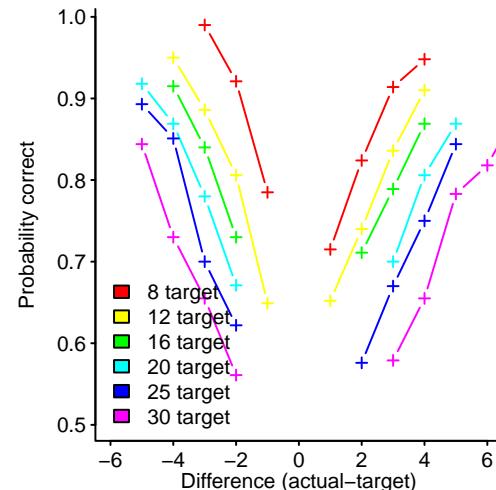


Figure 2.50: Probability a subject will successfully distinguish a difference between the number of dots displayed, and a specified target number (x-axis is the difference between these two values). Data extracted from van Oeffelen et al.<sup>1593</sup> [code](#)

What are the operating characteristics of the approximate number system? The characteristics that have most occupied researchers are the scale used (e.g., linear or logarithmic), the impact of number magnitude on cognitive performance, and when dealing with two numbers the effect of their relative difference in value on cognitive performance.<sup>385</sup>

Studies of the mental representation of single digit numbers<sup>388</sup> have found a linear scale used by subjects from western societies, and a logarithmic scale for subjects from indigenous cultures that have not had formal schooling.

Engineering and science sometimes deal with values spanning many orders of magnitude, a range that people are unlikely to encounter in everyday life. How do people mentally represent large value ranges?

A theoretical analysis,<sup>1247</sup> found that a logarithmic scale minimized representation error, based on the probability of a value occurring following a power-law distribution, assuming relative change is the psychologically-relevant measure, and that noise is present in the signal.

A study by Landy, Charlesworth and Ottmar<sup>913</sup> asked subjects to click the position on a line (labeled at the left end with one thousand and at the right end with one billion), which they thought was the appropriate location for each of the 182 values they saw (selected from 20 evenly spaced values between one thousand and one million, and 23 evenly spaced values between one million and one billion).

Various patterns occurred in subject responses; the top left plot in Figure 2.51 shows one of the most common. Most subjects placed one million at the halfway point (as-if using a logarithmic scale), placing values below/above a million on separate linear scales. Landy et al developed a model based on Category Adjustment theory,<sup>7</sup> where subjects selected a category boundary (e.g., one million, creating the categories: the thousands and the millions), a location for the category boundary along the line, and a linear(ish)<sup>ix</sup> mapping of values to relative position within their respective category.

Studying the learning and performance of simple arithmetic operations has proven complicated;<sup>511, 1595</sup> models of simple arithmetic performance<sup>931</sup> have been built. Working memory capacity has an impact on the time taken to perform mental arithmetic operations, and the likely error rate, e.g., remembering carry or borrow quantities during subtraction;<sup>749</sup> the human language used to think about the problem also has an impact.<sup>748</sup>

How do people compare multi-digit integer constants? For instance, do they compare them digit by digit (i.e., a serial comparison), or do they form two complete values before comparing their magnitudes (the so-called *holistic* model)? Studies show that the answer depends on how the comparisons are made, with results consistent with the digit by digit<sup>1699</sup> and holistic<sup>387</sup> approaches being found.

Other performance related behaviors include:

- *split effect*: taking longer to reject false answers that are close to the correct answer (e.g.,  $4 \times 7 = 29$ ), than those that are further away (e.g.,  $4 \times 7 = 33$ ),
- *associative confusion* effect: answering a different question from the one asked (e.g., giving 12 as the answer to  $4 \times 8 = ?$ , which would be true had the operation been addition),
- *plausibility judgments*:<sup>943</sup> using a rule, rather than retrieving a fact from memory, to verify the answer to a question; for instance, adding an odd number to an even number always produces an odd result,

Trial judges have been found<sup>1302</sup> to be influenced by the unit of measurement in sentencing (i.e., months or years), and to exhibit anchoring effects when deciding award damages.

## 2.7.1 Numeric preferences

Measurements of number usage, in general spoken and written form, show that people prefer to use certain values, either singly (sometimes known as *round numbers*) or as number pairs.

Number pairs (e.g., “10 to 15 hours ago”) have been found to follow a small set of rules,<sup>454</sup> including: the second number is larger than the first, the difference between the values is a divisor of the second value, and the difference is at least 5% of the second value.

<sup>ix</sup> Category Adjustment theory supports curvaceous lines.

A round number is any number commonly used to communicate an approximation of nearby values; round numbers are often powers of ten, divisible by two or five, and other pragmatic factors.<sup>772</sup> Round numbers can act as goals<sup>1262</sup> and as clustering points.<sup>1469</sup>

A usage analysis<sup>114</sup> shows that selecting a rounded interpretation yields the greater benefit, when there is a small chance that a rounded, rather than non-rounded, use occurred. If a speaker uses a round number, *round\_value*, the probability that the speaker rounded a nearby value to obtain it, is given by:

$$P(\text{Speaker\_rounded}|\text{round\_value}) = \frac{k}{k + \frac{1}{x} - 1}$$

where:  $k$  is the number of values likely to be rounded to *round\_value*, and  $x$  the probability that the speaker chooses to round. Figure 2.52 shows how the likelihood of rounding being used, increases rapidly as the number of possible rounded values increases.

A study by Basili, Panlilio-Yap, Ramsey, Shih and Katz<sup>110</sup> investigated the redesign and implementation of a software system. Figure 2.53 shows the number of change requests taking a given amount of time to decide whether the change was needed, and the time to design+implement the change. There are peaks at the round-numbers 1, 2 and 5, with 4 hours perhaps being the Parkinson's law target of a half day.

A study by King and Janiszewski<sup>846</sup> showed integer values between 1 and 100, in random order, to 360 subjects, asking them to specify whether they liked, disliked the number, or were neutral. Numbers ending in zero had a much higher chance of being liked, compared to numbers ending in other digits; see [developers/like-n-dis.R](#) for a regression model fitted to the like and dislike probability, based on the first/last digits of the number, along with the log of the value.

A study by van der Henst, Carles and Sperber<sup>1588</sup> approached people on the street, and asked them for the time; there is a 20% probability that the minutes were a multiple of five. When subjects were wearing an analogue watch, 98% of replies were multiples of five-minutes, when subjects were wearing digital watches the figure was 66%. Many subjects invested effort in order to communicate using a round number.

People sometimes denote approximate values using numerical expressions containing comparative and superlative qualifiers, such as “more than  $n$ ” and “at least  $n$ ”.

A study by Cummins<sup>345</sup> investigated the impact of number granularity on the range of values assigned by subjects, to various kinds of numerical expressions. Subjects saw statements of the form “So far, we’ve sold fewer than 60 tickets.” (in other statements fewer was replaced by more), and subjects were asked: “How many tickets have been sold? From ?? to ??, most likely ??”. Three numeric granularities were used: *coarse*, e.g., a multiple of 100, *medium* e.g., multiple of 10 and non-multiple of 100, and *fine* e.g., non-round such as 77.

The results found that the *most likely* value, given by subjects, was closest to the value appearing in the statement when it had a *fine* granularity and furthest away when it was *coarse*; see [reliability/CumminsModifiedNumeral.R](#). Figure 2.54 shows the “From to” range given by each subject, along with their best estimate (in green) for statements specifying “less than 100” and “more than 100”.

When specifying a numeric value, people may also include information that expresses their uncertainty, using a so called *hedge* word, e.g., “about”, “around”, “almost”, “almost exactly”, “at least”, “below” and “nearly”.

A study by Ferson, O’Rawe, Antonenko, Siegrist, Mickley, Luhmann, Sentz and Finkel<sup>492</sup> investigated interpretations of numerical uncertainty. Subjects were asked to specify minimal and maximal possible values, for their interpretation of the quantity expressed in various statements, such as: “No more than 100 people.”

Figure 2.55 shows the cumulative probability of the relative uncertainty of numeric phrases (calculated as:  $\frac{\text{minimal}-\text{actual}}{\text{minimal}}$  and  $\frac{\text{maximal}-\text{actual}}{\text{maximal}}$ ), for the hedge words listed in the legends.

The relative size of objects being quantified has been found to have an effect on the interpretation given to quantifiers.<sup>1150</sup>

emailed...?

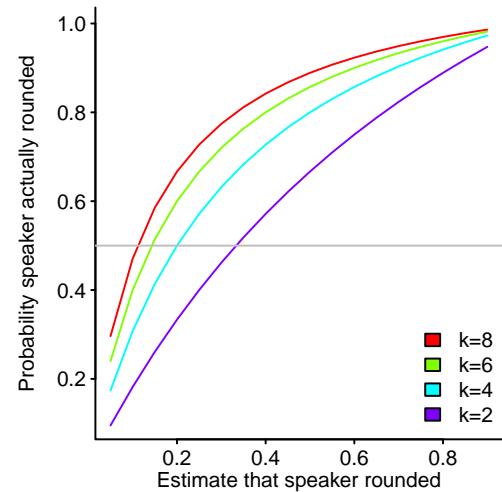


Figure 2.52: Probability the rounded value given has actually been rounded, given an estimate of the likelihood of rounding, and the number of values likely to have been rounded; grey line shows 50% probability of rounding. [code](#)

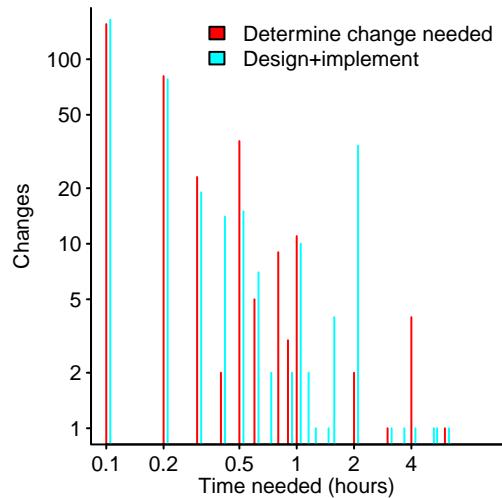


Figure 2.53: Number of change requests having a given recorded time to decide whether change needed, and time to implement. Data from Basili et al.<sup>110</sup> [code](#)

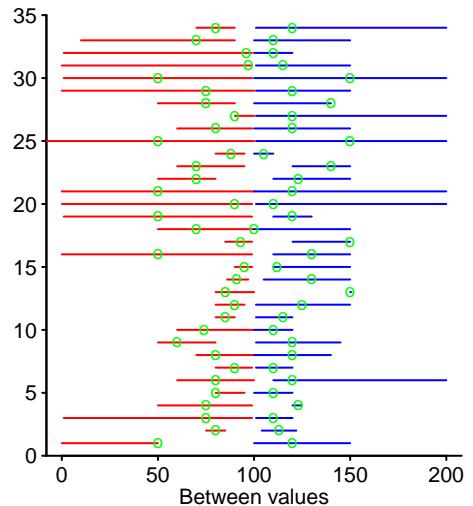


Figure 2.54: Min/max range of values (red/blue lines), and best value estimate (green circles), given by subjects interpreting the value likely expressed by statements containing “less than 100” and “more than 100”. Data kindly provided by Cummins.<sup>345</sup> [code](#)

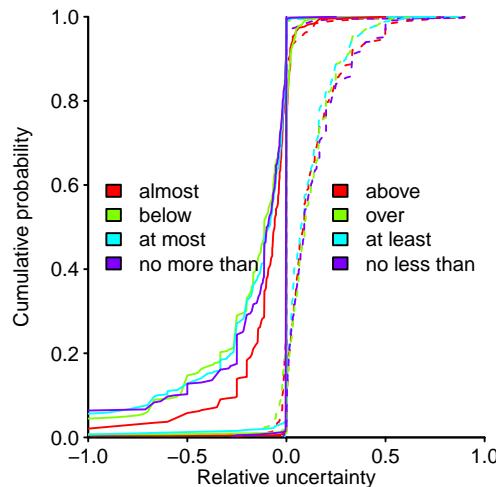


Figure 2.55: The cumulative probability of subjects expressing a given relative uncertainty, for numeric phrases using given hedge words. Data kindly provided by Fer son.<sup>492</sup> code

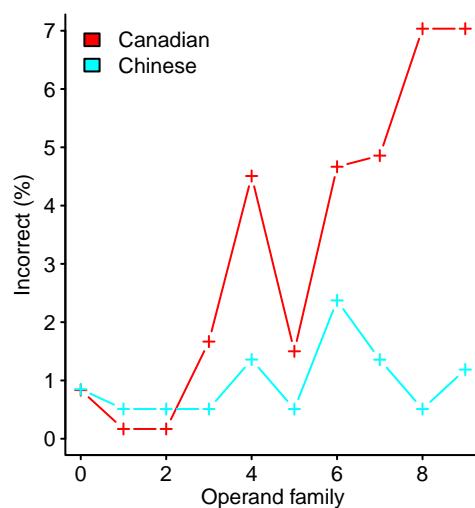


Figure 2.56: Percentage of incorrect answers to arithmetic problems, given by Canadian and Chinese students, for each operand family value. Data kindly provided by LeFevre.<sup>939</sup> code

## 2.7.2 Symbolic distance and problem size effect

When people compare two items sharing a distance-like characteristic, the larger the distance between two items, the faster people are likely to respond to a comparison question involving this characteristic (e.g., comparisons of social status<sup>288</sup> and geographical distance;<sup>705</sup> also see Figure 2.18); this is known as *symbolic distance effect*. Inconsistencies between the symbolic distance and actual distance, can increase the error rate,<sup>680</sup> e.g., is the following relation true?  $3 > 5$ .

In a study by Tzelgov, Yehene, Kotler and Alon,<sup>1575</sup> subjects trained one-hour per day for six days, learning the relative order of nine graphical symbols. A symbolic distance effect was seen in subject performance, when answering questions about relative graphical symbol order.

Studies have found that the time taken to solve a simple arithmetic problem, and the error rate, increase as the value of both operands increases (e.g., subjects are slower to solve  $9 + 7$ , than  $2 + 3$ ,<sup>244</sup> see [developers/campbell1997.R](#)); this is known as the *problem size effect*. However, the characteristics of student performance can vary widely.

A study by LeFevre and Liu<sup>939</sup> investigated the performance of Canadian and Chinese university students on simple multiplication problems (e.g.,  $7 \times 4$ ). Figure 2.56 shows the percentage error rate for problems containing values from a given operand family (e.g.,  $2 \times 6$  contains values from the 2 and 6 operand family, and an incorrect answer to this problem counts towards both operand families). The hypothesis for the difference in error rate was student experience; during their education Canadian students were asked to solve more multiplication problems containing small values, compared to large values; an analysis of the work-books of Chinese students found the opposite distribution of operand values, i.e., Chinese students were asked to solve more problems involving large operands, compared to small operands.

## 2.7.3 Estimating event likelihood

The ability to detect regularities in the environment, and to take advantage of them is a defining characteristic of intelligent behavior.

In some environments, the cost of failing to correctly detect an object or event may be significantly higher than the cost of acting as-if an event occurred, when none occurred. People appear to be hardwired to detect some patterns, e.g., seeing faces in a random scattering of objects.

People sometimes have expectations of behavior for event sequences. For instance, an expectation that sequences of random numbers have certain attributes,<sup>39,466</sup> e.g., frequent alternation between different values (which from a mathematical perspective, are of form of regularity).

People tend to overestimate the likelihood of uncommon events and underestimate the likelihood of very common events. A variety of probability weighting functions have been proposed to explain the experimental evidence.<sup>590</sup>

The log-odds of the proportion estimated,  $\log \frac{p_e}{1-p_e} \Rightarrow lo_{pe}$  (where  $p_e$  is the proportion estimated), is given by:

$$lo_{pe} = \gamma lo_{pa} + (1 - \gamma) \delta$$

where:  $\gamma$  is a constant (interpreted as a relative weight on the perception of  $lo_{pa}$ , the log-odds of the actual proportion), and  $\delta$  is a constant (interpreted as a prior expectation).<sup>x</sup>

Figure 2.57 shows data from various surveys of public perception of demographic questions involving the percentage of a population containing various kinds of people, along with a fitted log-odds proportional model (grey line shows estimated equals actual).

The probability of a particular event occurring may not be fixed, the world is constantly changing, and the likelihood of an event may change. Studies have found that people are responsive to changes in the frequency with which an event occurs,<sup>540</sup> and to changes in the available information.<sup>936</sup>

<sup>x</sup> This equation is sometimes written using  $p_a$  directly, rather than using log-odds, i.e.,  $p_e = \frac{\delta^{1-\gamma} p_a^\gamma}{\delta^{1-\gamma} p_a^\gamma + (1-p_a)^\gamma}$ .

A change in the occurrence of an event can result in a change to what is considered to be an occurrence of the event. A study by Levari, Gilbert, Wilson, Sievers, Amodio and Wheatley<sup>946</sup> showed subjects a series of randomly colored dots, one at a time (the colors varied on a continuous scale, between purple and blue); subjects were asked to judge whether each dot they saw was blue (each subject saw 1,000 dots). Subjects were divided into two groups: for one group the percentage of blue dots did not change over time, while for the second group the percentage of blue dots was decreased after a subject had seen 200 dots.

Figure 2.58 is based on data from the first and last series of 200 dots seen by each subject; the x-axis is an objective measure of the color of each dot, the y-axis is the percentage of dots identified as blue (averaged over all subjects). The responses for subjects in the constant blue group did not change between the first/last 200 dots (red and green lines). For the decreased blue group, after subjects experienced a decline in the likelihood of encountering a blue dot, the color of what they considered to be a blue dot shifted towards purple (blue and purple lines).

In later experiments subjects were told that the prevalence of blue dots "might change", and would "definitely decrease"; the results were unchanged.

## 2.8 High-level functionality

This section discusses high-level cognitive functionality that has an impact on software development, but for which there is insufficient material for a distinct section.

### 2.8.1 Personality & intelligence

Perhaps the most well-known personality test is the *Meyer-Briggs type indicator*, or MBTI (both registered trademarks of Consulting Psychologists Press). The reliability and validity of this test has been questioned,<sup>1040, 1253</sup> with commercial considerations, and a changing set of questions making research difficult. The International Personality Item Pool (IPIP)<sup>585</sup> is a public domain measure, which now has available samples containing hundreds of thousands of responses.<sup>777</sup>

The Five-Factor model<sup>584</sup> structures personality around five basic traits: neuroticism, extroversion, openness, agreeableness, and conscientiousness; it has its critics.<sup>1411</sup> Studies that have attempted to identify personality types, based on these five traits have suffered from small sample size (e.g., 1,000 people), but studies based on IPIP data, using samples containing 100,000 to 500,000 responses, claim to have isolated four personality types.<sup>559</sup>

Tests measuring something called IQ have existed for over 100 years. The results are traditionally encapsulated in a single number, but tests claiming to measure the various components of intelligence are available;<sup>375</sup> the model is that people possess a general intelligence  $g$ , plus various domain specific intelligences, e.g., in tests involving maths the results would depend on  $g$  and maths specific intelligence, and in tests involving use of language the results would depend on  $g$  and language specific intelligence.

Cultural intelligence hypothesis, specific set of social skills for exchanging knowledge in social groups children and chimpanzees have similar cognitive skills for dealing with the physical world, but children have more sophisticated skills for dealing with the social world.<sup>690</sup>

Without reliable techniques for measuring personality traits, it is not possible to isolate characteristics likely to be beneficial or detrimental to software development. Perhaps one of the most important traits is an ability to concentrate for large amounts of time on particular activities.

emailed asking for data, may be on its way...??

### 2.8.2 Risk taking

Making a decision based on incomplete or uncertain information involves an element of risk. How do people integrate risk into their decision-making process?

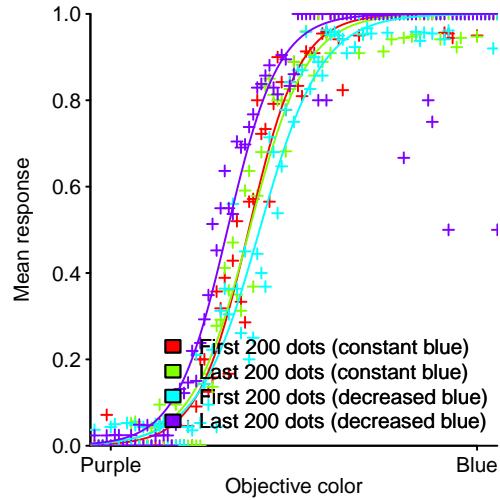


Figure 2.58: Mean likelihood that a subject considered a dot of a given color to be blue, for the first/last 200 dots seen by two groups of subjects. Data from Levari et al.<sup>946</sup> code

The term *risk asymmetry* refers to the fact that people have been found to be *risk averse* when deciding between alternatives that have a positive outcome, but are *risk seeking* when deciding between alternatives that have a negative outcome.<sup>xi</sup>

While there is a widespread perception that women are more risk averse than men, existing studies are either not conclusive or show a small effect<sup>494</sup> (many suffer from small sample sizes, and dependencies on the features of the task subjects are given). For the case of financial risks, the evidence<sup>276</sup> that men are more willing to take financial risks than women is more clear cut. The evidence from attempts to improve road safety is that “protecting motorists from the consequences of bad driving encourages bad driving”.<sup>7</sup>

A study by Jones<sup>791</sup> investigated the possibility that some subjects, in an experiment involving recalling information about previously seen assignment statements, were less willing to risk giving an answer, when they had opportunity to specify that in real-life, they would refer back to previously read code. Previous studies<sup>786,787</sup> had found that a small percentage of subjects consistently gave much higher rates of “would refer back” answers. One explanation is that these subjects had a smaller short term memory capacity than other subjects (STM capacity does vary between people), another is that these subjects are much more risk averse than the other subjects.

The Domain-Specific Risk-Taking (DOSPERT) questionnaire<sup>171,1645</sup> was used to measure subject’s risk attitude. The results found no correlation between risk attitude (as measured by DOSPERT), and number of “would refer back” responses.

### 2.8.3 Decision-making

A distinction is commonly made between decisions made under risk, and decisions made under uncertainty.<sup>1618</sup> Decisions are made under risk when all the information about the probability of the outcomes is available, and one of the available options has to be selected. Many of the studies investigating human decision-making, provide subjects with all the information about the probability of the outcomes and ask them to select an option, i.e., they involve decision under risk.

Decisions are made under uncertainty when the available information is incomplete and possibly inaccurate. Human decision-making often takes place in an environment of incomplete information and limited decision-making resources (e.g., working memory capacity and thinking time); people have been found to adopt various strategies to handle this situation,<sup>1017</sup> balancing the predicted cognitive effort required to use a particular decision-making strategy against the likely accuracy achieved by that strategy.

The term *bounded rationality*<sup>1444</sup> is used to describe an approach to problem solving applied when limited cognitive resources are available. A growing number of studies<sup>570</sup> have found that the methods used by people to make decisions and solve problems are often close enough to optimal<sup>xii</sup>, given the resources available to them; even when dealing with financial matters.<sup>1044</sup> If people handle money matters in this fashion, their approach to software development decisions is unlikely to fare any better.

A so-called *irregular choice* occurs when a person who chooses B from the set of items {A, B, C} does not choose from the set {A, B}; irregular decision makers have been found<sup>1543</sup> to be more common among younger (18–25) subjects, but less are common with older (60–75) subjects.

Consumer research into understanding how shoppers decide among competing products uncovered a set of mechanisms that are applicable to decision-making in general, e.g., decision-making around the question of which soap will wash the whitest is no different from the question of whether an **if** statement, or a **switch** statement be used. Before a decision can be made, a decision-making strategy has to be selected. People have been found to use a number of different decision-making strategies.<sup>1222</sup>

Decision strategies differ in several ways, for instance, some make trade-offs among the attributes of the alternatives (making it possible for an alternative with several good attributes

---

<sup>xi</sup> While studies<sup>686</sup> based on subjects drawn from non-WEIRD societies sometimes produce different results, this book assumes developers are WEIRD.

<sup>xii</sup> Some researchers interpret bounded rationality as human decision-making producing results as-if people optimize under cognitive constraints, while others<sup>566</sup> don’t require people to optimize, but to use algorithms that produce results that are good enough.

to be selected instead of the alternative whose only worthwhile attribute is excellent); they also differ in the amount of information that needs to be obtained, and the amount of cognitive processing needed to make a decision. Named decision-making strategies include: the weighted additive rule, the equal weight heuristic, the frequency of good and bad features heuristic, the majority of confirming dimensions heuristic, the satisficing heuristic the lexicographic heuristic, the habitual heuristic and in recent years decision by sampling.<sup>1507</sup>

There is a hierarchy of responses for how people deal with time pressure:<sup>1222</sup> they work faster; if that fails, they may focus on a subset of the issues; if that fails, they may change strategies (e.g., from alternative based to attribute based).

**Copying others:** Find good solutions to problems is hard, doing what others do can be a cost effective strategy. *When Strategies*: copy when established behavior is unproductive, copy when asocial learning is costly, copy when uncertain. *Who Strategies*: copy the majority, copy if rare, copy successful individuals, copy if better, copy if dissatisfied, copy good social learners, copy kin, copy "friends", copy older individuals.

Studies have found that having to justify a decision can affect the choice of decision-making strategy used.<sup>1545</sup> One strategy that handles accountability is to select the alternative that the perspective audience is thought most likely to select.<sup>1544</sup> People who have difficulty determining, which alternative has the greatest utility tend to select the alternative that supported the best overall reasons (for choosing it).<sup>1446</sup>

Requiring developers to justify why they have not followed existing practice can be a two-edged sword. Developers can respond by deciding to blindly follow everybody else (a path of least resistance), or they can invest effort in evaluating alternatives (not necessarily cost effective behavior, since the invested effort may not be warranted by the expected benefits). The extent to which some people will blindly obey authority was chillingly demonstrated in a number of studies by Milgram.<sup>1077</sup>

Social pressure can cause people to go along with decisions voiced by those involved in the decision process (i.e., social conformity as a signal of belonging,<sup>161</sup> such as corporate IT fashion Figure 1.15). A study by Asch<sup>62</sup> asked groups of seven to nine subjects to individually state, which of three black strips they considered to be the longest (see Figure 2.59), the group sat together in front of the stripes, and subjects could interact with each other; all the subjects, except one, were part of the experiment, and in 12 of 18 questions selected a stripe that was clearly not the longest (i.e., the majority gave an answer clearly at odds with visible reality). It was arranged that the actual subject did not have to give an answer until hearing the answers of most of the other subjects.

The actual subject, in 24% of groups, always gave the correct answer; in 27% of groups the subject agreed with the incorrect majority answer between eight and twelve times, and just under half varied in the extent to which they followed the majority decision. When the majority selected the most extreme incorrect answer, i.e., the shortest stripe, subjects giving an incorrect answer selected the less extreme incorrect answer in 20% of cases.

How a problem is posed can have a large impact on the decision made.

A study by Regnell, Höst, och Dag, Beremark and Hjelm<sup>1319</sup> asked 10 subjects to assign a relative priority to two lists of requirements (subjects' had a total budget of 100,000 units and had to assign units to requirements). One list specified that subjects should prioritize the 17 high level requirements, and the other list specified that a more detailed response, in the form of prioritizing every feature contained within each high level requirement, be given.

Comparing the totals for the 17 high level requirements (summing the responses for the detailed list), showed that the correlation was not very strong (the mean across 10 subjects was 0.46, see [projects/prioritization.R](#)).

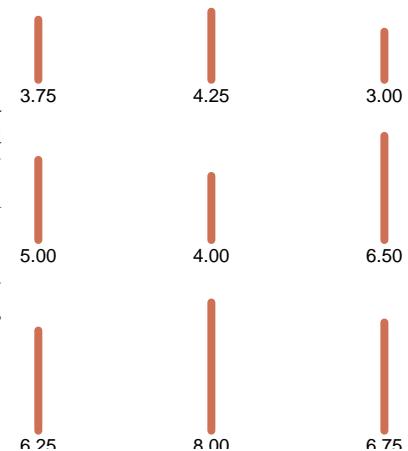


Figure 2.59: Each row shows a scaled version of the three stripes, along with actual lengths in inches, from which subjects were asked to select the longest. Based on Asch.<sup>62</sup> code

## 2.8.4 Expected utility and Prospect theory

The outcome of events is often uncertain. If events are known to occur with probability  $p_i$ , with each producing a value of  $X_i$ , then the expected outcome value is given by:

$$E[X] = p_1X_1 + p_2X_2 + p_3X_3 + \dots + p_nX_n$$

For instance, given a 60% chance of winning £10 and a 40% chance of winning £20, the expected winnings are:  $0.6 \times 10 + 0.4 \times 20 = 14$

When comparing the costs and benefits of an action, decision makers often take into account information on their current wealth, e.g., a bet offering a 50% chance of winning £1 million, and a 50% chance of losing £0.9 million has an expected utility of £0.05 million; would you take this bet, unless you were very wealthy? Do you consider £20 to be worth twice as much as £10?

The mapping of a decision's costs and benefits, to a decision maker's particular circumstances, is made by what is known as a *utility function*,  $u$ ; the above equation becomes (where  $W$  is the decision maker's current wealth):

$$E[x] = p_1 u(W + X_1) + p_2 u(W + X_2) + p_3 u(W + X_3) + \cdots + p_n u(W + X_n)$$

In some situations a decision maker's current wealth might be effectively zero, e.g., they have forgotten their wallet, or because they have no authority to spend company money on work related decisions (personal wealth of employees is unlikely to factor into many of their work decisions).

Figure 2.60 shows possible perceived utilities of an increase in wealth. A risk neutral decision maker perceives the utility of an increase in wealth as being proportional to the increase (green,  $u(w) = w$ ), a risk loving decision maker perceives the utility of an increase in wealth as being proportionally greater than the actual increase (red,  $u(w) = w^2$ ), while a risk averse decision maker perceives the utility of an increase having proportionally less utility (blue,  $u(w) = \sqrt{w}$ ).

A study by Kina, Tsunoda, Hata, Tamada and Igaki<sup>845</sup> investigated the decisions made by subjects (20 open source developers) when given a choice between two tools, each having various probabilities of providing various benefits, e.g., *Tool*<sub>1</sub> which always saves 2 hours of work, or *Tool*<sub>2</sub> which saves 5 hours of work with 50% probability, and has no effect on work time with 50% probability.

Given a linear utility function, *Tool*<sub>1</sub> has an expected utility of 2 hours, while *Tool*<sub>2</sub> has an expected utility of 2.5 hours. The results showed 65% of developers choosing *Tool*<sub>1</sub>, and 35% choosing *Tool*<sub>2</sub>: clearly those 65% were not using a linear utility function; use of a square-root utility function would produce the result seen:  $1 \times \sqrt{2} > 0.5 \times \sqrt{5} + 0.5 \times \sqrt{0}$ .

Prospect theory...

## 2.8.5 Overconfidence

While overconfidence can create unrealistic expectations, and lead to hazardous decisions being made (e.g., to allocate insufficient resources to complete a job), simulation studies<sup>776</sup> have found that overconfidence has benefits in some situations, averaged over a population (see [developers/overconfidence/0909.R](#)). A study<sup>63</sup> of commercialization of new inventions, found that while inventors are significantly more confident and optimistic than the general population, the likely return on their investment of time and money in their invention is negative; a few receive a huge pay-off.

A study by Lichtenstein and Fishhoff<sup>955</sup> asked subjects general knowledge questions, with the questions divided into two groups, hard and easy. Figure 2.61 shows that subjects' overestimated their ability (x-axis) to correctly answer hard questions, but underestimated their ability to answer easy questions; green line denotes perfect self-knowledge.

These, and subsequent results, show that the skills and knowledge that constitute competence in a particular domain, are the same skills needed to evaluate one's (and other people's) competence in that domain. *Metacognition* is the term used to denote the ability of a person to accurately judge how well they are performing.

Peoples' belief in their own approach to getting things done can result in them ignoring higher performing alternatives;<sup>53</sup> this behavior has become known as *the illusion of control*.<sup>916</sup>

It might be thought that, people who have previously performed some operation, would be in a position to make accurate predictions about future performance on those operations. However, studies have found<sup>1356</sup> that, while people do use their memories of the duration of past events, to make predictions of future event duration, their memories are systematic underestimates of past duration. People appear to underestimate future event duration because they underestimate past event duration.

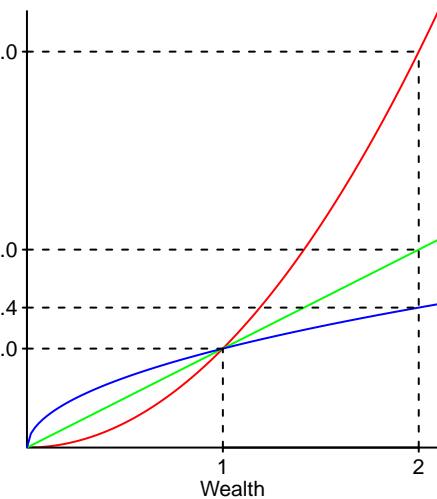


Figure 2.60: Risk neutral (green,  $u(w) = w$ ), risk loving (red, quadratic), and risk averse (blue, square-root) utility functions. [code](#)

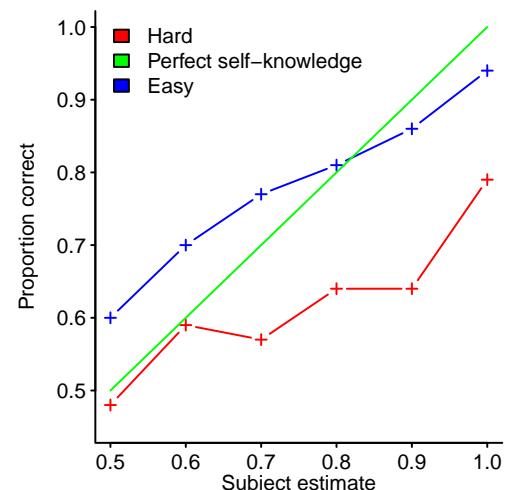


Figure 2.61: Subjects' estimate of their ability (x-axis) to correctly answer a question and actual performance in answering on the left scale. The responses of a person with perfect self-knowledge is given by the green line. Data extracted from Lichtenstein et al.<sup>955</sup> [code](#)

## 2.8.6 Time discounting

People seek to consume pleasurable experiences sooner, and to delay their appointment with painful experiences, i.e., people tend to accept less satisfaction in the short-term, than could be obtained by pursuing a longer-term course of action; a variety of models have been proposed.<sup>413</sup> Studies have found that animals, including humans, appear to use a hyperbolic discount function for time variable preferences.<sup>391</sup> The hyperbolic delay discount function is:

$$v_d = \frac{V}{1+kd}$$

where:  $v_d$  is the delayed discount,  $V$  the undiscounted value,  $d$  the delay and  $k$  some constant.

A property of this hyperbolic function is that curves with different values of  $V$  and  $d$  can cross. In the example Figure 2.62, the perceived present value of two future rewards (red and blue lines) starts with red being initially greater than blue, as time passes (i.e., the present time moves right, and the delay before receiving the rewards decreases) the reward denoted by the blue line out-grows the red line, until there is a reversal in perceived present value. When both rewards are far in the future, the larger amount has the greater perceived value, studies have found<sup>852</sup> that subjects switch to giving a higher value to the lesser amount as the time of receiving the reward gets closer.

A study by Becker, Fagerholm, Mohanani and Chatzigeorgiou<sup>126</sup> asked professional developers how many days of effort would need to be saved over the lifetime of a project, to make it worthwhile investing time integrating a new library, compared to spending that time implementing a feature in the project backlog. The developers worked at companies who developed and supported products over many years, and were asked to specify saving for various project lifetimes.

Figure 2.63 shows the normalised savings specified by developers from one company, for given project lifetimes. Many of the lines are concave, showing that these developers are applying a decreasing interest rate, for the time investment, as project lifetime increased (if a fixed interest rate was applied, the lines would curve up). fitting a hyperbolic model...

## 2.8.7 Developer performance

Companies seek to hire those people who will give the best software development performance. Currently, the only reliable method of evaluating developer performance is by measuring developer outputs (a good enough model of the workings of human mental operations remains in the future).

One operational characteristic of the brain that can be estimated is the number of operations that could potentially be performed per second (a commonly used method of estimating the performance of silicon-based processors).

The brain might simply be a very large neural net, so there may be no instructions to count as such; Merkle<sup>1068</sup> used the following approaches to estimate the number of synaptic operations per second (the supply of energy needed to fire neurons limits the number that can be simultaneously active, in a local region, to between 1% and 4% of the neurons in that region<sup>944</sup>):

- multiplying the number of synapses ( $10^{15}$ ), by their speed of operation (about 10 impulses/second), gives  $10^{16}$  synapse operations per second (if the necessary energy could be delivered to all of them at the same time),
- the retina of the eye performs an estimated  $10^{10}$  analog add operations per second. The brain contains  $10^2$  to  $10^4$  times as many nerve cells as the retina, suggesting that it can perform  $10^{12}$  to  $10^{14}$  operations per second,
- the brain's total power dissipation of 25 watts (an estimated 10 watts of useful work), and an estimated energy consumption of  $5 \cdot 10^{-15}$  joules for the switching of a nerve cell membrane, provides an upper limit of  $2 \cdot 10^{15}$  operations per second.

A synapse switching on and off is the same as a transistor switching on and off, in that they both need to be connected to other switches to create a larger functional unit. It is not known

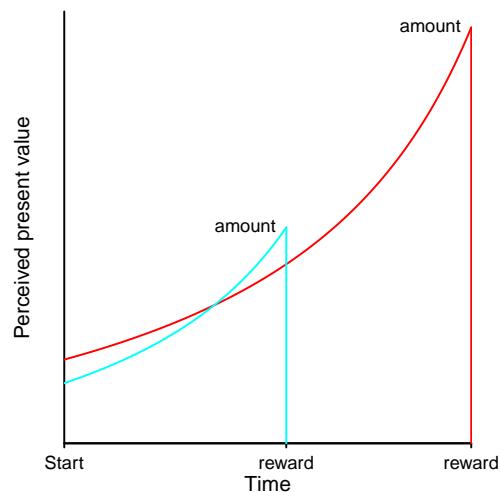


Figure 2.62: Perceived present value (moving through time to the right) of two future rewards. [code](#)

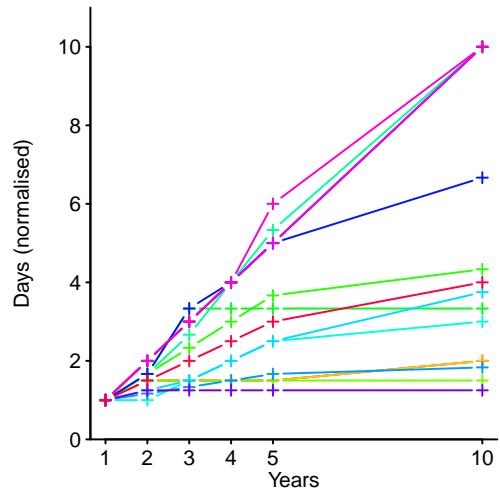


Figure 2.63: Saving required (normalised), over a project having a given duration, before subjects would make long term investment. Data from Becker et al.<sup>126</sup> [code](#)

how many synapses are used to create functional units, or even what those functional units might be. The distance between synapses is approximately 1 mm, and sending a signal from one part of the brain to another part requires many synaptic operations, for instance, to travel from the front to the rear of the brain requires at least 100 synaptic operations to propagate the signal. So the number of synaptic operations per high-level, functional operation, is likely to be high. Silicon-based processors can contain millions of transistors; the potential number of transistor-switching operations per second might be greater than  $10^{14}$ , but the number of instructions executed is significantly smaller.

Although there have been studies of the information-processing capacity of the brain (e.g., visual attention<sup>1602</sup> and storage rate into long-term memory<sup>195,909</sup>), we are a long way from being able to deduce the likely work rates of the components of the brain while performing higher level cognitive functions.

Processing units need a continuous supply of energy to function. The brain does not contain any tissue that stores energy, and obtains all its energy needs through the breakdown of blood-borne glucose. Consuming a glucose drink has been found to increase blood glucose levels, and enhance performance on various cognitive tasks.<sup>834</sup> Also, fluctuations in glucose levels have an impact on an individual's ability to exert self-control,<sup>536</sup> with some glucose intolerant individuals not always acting in socially acceptable ways.<sup>xiii</sup>

How do developers differ in their measurable output performance?

Although much talked about, there has been little research on individual developer productivity. One review<sup>742</sup> of studies of employee output variability, found a progression of standard deviation, about the mean, from 19% to 48%, as job complexity increased (not software related). Claims of a 28-to-1 productivity difference between developers, is sometimes still bandied about. The, so called *Grant-Sackman study*<sup>614</sup> is based on an incorrect interpretation of a summary of their experimental data.<sup>1277</sup> The data shows a performance difference of around 6-to-1 between developers using batch vs. online, for creating software (see [group-compare/GS-perm-diff.R](#) and Figure 8.22).

Most organizations do not attempt to measure the mental characteristics of developer job applicants; unlike many other jobs where individual performance is an important consideration. Whether this is because of the existing non-measurement culture, lack of reliable measuring procedures, or fear of frightening off prospective employees is not known.

A study of development and maintenance costs of programs written in C and Ada<sup>1695</sup> found no correlation between salary grade (or employee rating), and rate of bug fix or feature implementation rate.

One metric used in software testing is number of faults found. In practice non-failing tests, written by software testers, are useful because they provide evidence that particular functionality behaves as expected.

A study by Iivonen<sup>747</sup> analysed the defect detection performance of those involved in testing software at several companies. Table 2.8 shows the number of defects detected by six testers (all but the first column, show percentages), along with self-classification of the seriousness, followed by the default status assigned by others.

Tester	Defects	Extra Hot	Hot	Normal	Open	Fixed	No fix	Duplicate	Cannot reproduce
A	74	4	1	95	12	62	26	12	0
B	73	0	56	44	15	87	6	2	5
C	70	0	29	71	36	71	24	0	4
D	51	0	27	73	33	85	6	0	9
E	50	2	16	82	30	89	9	0	3
F	18	0	22	78	22	64	14	0	21

Table 2.8: Defects detected by six testers (some part-time and one who left the company during the study period), and their status. Data from Iivonen.<sup>747</sup>

A performance comparison, based on defects reported, requires combining these figures (and perhaps others, e.g., likelihood of being experienced by a customer) into a value that can be reliably compared across testers. Defects differ in their commercial importance, and a relative

<sup>xiii</sup> There is a belief in software development circles that consumption of chocolate enhances cognitive function. A systematic review of published studies<sup>1388</sup> found around a dozen papers addressing this topic, with three finding some cognitive benefits and five finding some improvement in mood state.

weight for each classification has to be decided; should the weight of “No fix” be larger than that given to “Cannot reproduce” or “Duplicate”?

To what extent would a tester’s performance, based on measurements involving one software system in one company, be transferable to another system in the same company or another company? Iivonen interviewed those involved in testing, to find out what characteristics were thought important in a tester. Knowledge of customer processes and use cases, was a common answer; this usage knowledge enables testers to concentrate on those parts of the software that customers are most likely to use and be impacted by incorrect operation, it also provides the information needed to narrow down the space of possible input values.

Knowledge of the customer ecosystem and software development skills are the two blades, in Figure 2.1, that have to mesh together to create useful software systems.

??

## 2.8.8 Miscellaneous

Research in human-computer interaction investigates a variety of human performance characteristics; some of these characteristics include physical movement (e.g., hand or eye movement), not mental operations. The equations fitted to experimental performance data, for some of these characteristics, often contain logarithms, and attention has been drawn to the similarity of form to the equations used in information theory.<sup>1003</sup> The use of a logarithmic scale, to quantify the perception of stimuli, minimizes relative error.<sup>1267</sup> Some of the more commonly encountered performance characteristics include:

- Fitts’ law: time taken,  $RT$ , to move a distance  $D$ , to an object having width  $W$ , is  $RT = a + b \log \left[ \frac{2D}{W} \right]$ , where  $a$  and  $b$  are constants. In deriving this relationship, Fitts drew on ideas from information theory, and used a simplified version of Shannon’s law; the unsimplified version implies:  $RT = c + d \log \left[ \frac{D+W}{W} \right]$ ,<sup>1003</sup>
- Hick’s law: time taken,  $RT$ , to choose an item from a list of  $K$  items, is  $RT = a + b \log(K)$ , where  $a$  and  $b$  are constants;  $a$  is smaller for humans than pigeons.<sup>1609</sup> A study by Hawkins, Brown, Steyvers and Wagenmakers<sup>667</sup> displayed a number of squares on a screen, and asked subjects to select the square whose contents had a particular characteristic. Figure 2.64 shows how subject response time increased (and accuracy decreased), as the log of the number of choices. A different color is used for each of the 36 subjects, with colors sorted by performance on the two-choice case. This study found that problem context could cause subjects to make different time/accuracy performance trade-offs,
- Ageing effects: the Seattle Longitudinal Study<sup>1381</sup> has been following the intellectual development of over six thousand people since 1956 (surveys of the individuals in the study are carried out every seven years). Findings include: “ . . . there is no uniform pattern of age-related changes across all intellectual abilities, . . . ”, and “ . . . reliable replicable average age decrements in psychometric abilities do not occur prior to age 60, but that such reliable decrements can be found for all abilities by age 74 . . . ” An analysis<sup>184</sup> of the workers on the production line at a Mercedes-Benz assembly plant found that productivity did not decline until at least up to age 60.

Years of experience vs. experimental performance....

Changes in a person’s social and economic standing over time are likely to have a much larger impact than changes in mental ability. For instance, commitments outside work (e.g., family) reduce the time available for acquiring new skills; people who are paid more (an individual’s salary is likely to increase over time) are expected to deliver more (people promoted to their maximum level of incompetence; its cheaper to hire younger people who are already familiar with new technologies than retrain more expensive older people). and the data backing up these claims...

See Figure 8.13 for an age distribution of developers,

- the *sunk cost effect* is the tendency to persist in an endeavour, once an investment of time, or money, has been made. This effect is observed in other animals,<sup>1138</sup> and so presumably this behavior provides survival benefits.

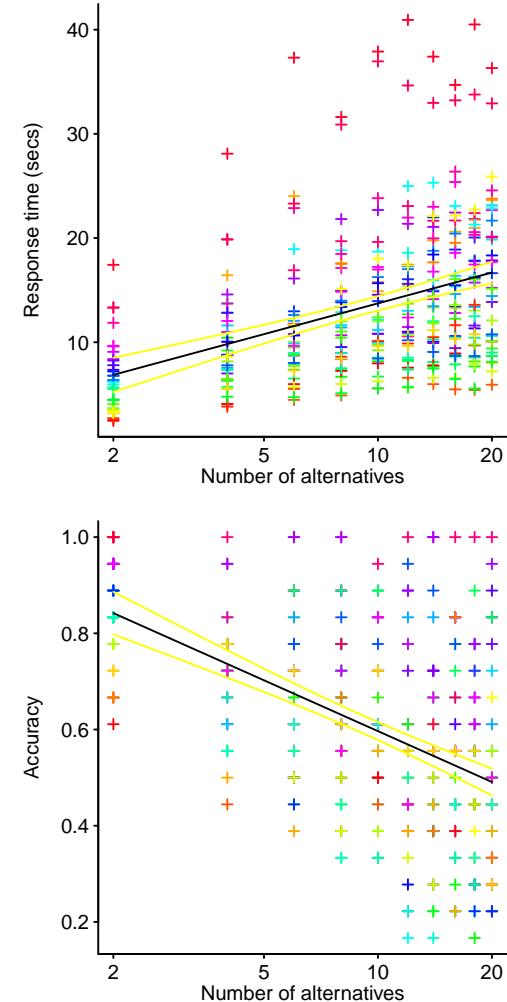


Figure 2.64: Mean time for each of 36 subjects to choose between a given number of alternatives (upper), and accuracy rate for a given number of alternatives (lower); lines are regression fits (yellow is 95% confidence interval), and color used for each subject sorted by performance on the two-choice case. Data from Hawkins et al.<sup>667</sup> code



# Chapter 3

## Cognitive capitalism

### 3.1 Introduction

Software systems are intangible goods that are products of cognitive capitalism; human cognition is the means of production.

The major motivations for an individual to be involved in the production of software are money and hedonism. People might be motivated to write software by the salary they are paid, by the owners of an organization that employs them, or they might be motivated by non-salary income (of an individual's choosing), e.g., enjoyment from working on software systems, scratching an itch, being involved in a social activity, etc.

Motivation may not have any effect on the work environment, in which software is produced. To make the cognitariate feel happy and fulfilled, spending their cognitive capital striving to achieve management goals, companies have industrialised Bohemia.

While salary based production is likely to be distributed under a commercial license, and hedonism based production under some form of open source<sup>i</sup> license, this is not always the case.

This chapter discusses cognitive capitalism using the tools of economic production, which deals with tradeable quantities, such as money, time and pleasure. Many of the existing capitalist structures are oriented towards the production of tangible goods, and are slowly being adapted to deal with the growing market share of intangible goods.<sup>142,658</sup>

This book is oriented towards the producers of software, rather than its consumers, e.g., it focuses on maximizing the return on investment for producers of software.

Human characteristics that affect cognitive performance are the subject of Chapter 2.

The sector economic model groups human commercial activities into at least three sectors:<sup>833</sup> the primary sector produces or extracts raw materials, e.g., agriculture, fishing and mining, the secondary sector processes raw materials, e.g., manufacturing and construction, and the tertiary sector provides services. The production of intellectual capital is sometimes referred to as the quaternary sector. Figure 3.1 shows the percentage of the US workforce employed in the three sectors over the last 160 years (plus government employment).

How much money is spent on software production?

A study by Parker and Grimm<sup>1212</sup> investigated business and government expenditure on software in the U.S.A. They divided software expenditure into two categories: *custom software*, as software tailored to the specifications of an organization, and *own-account software* which consists of in-house expenditures for new or significantly-enhanced software created by organizations for their own use. Figure 3.2 shows annual expenditure from 1959 to 1998, by US businesses (plus lines), and the US federal and state governments (smooth lines).

Figure 3.3 shows the growth in the number of people employed by some major software companies...

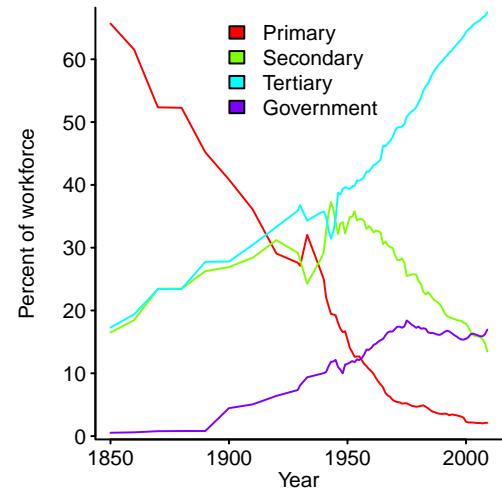


Figure 3.1: Percentage of employment by US industry sector 1850-2009. Data kindly provided by Kossik.<sup>880</sup> [code](#)

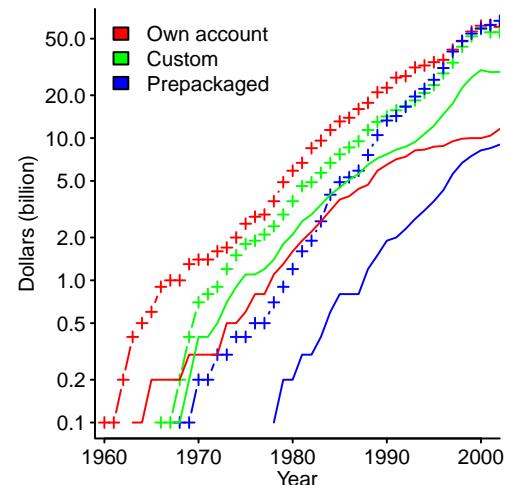


Figure 3.2: Annual expenditure on custom, own account and prepackaged software by US business (plus lines) and the US federal and state governments (smooth lines). Data from Parker et al.<sup>1212</sup> [code](#)

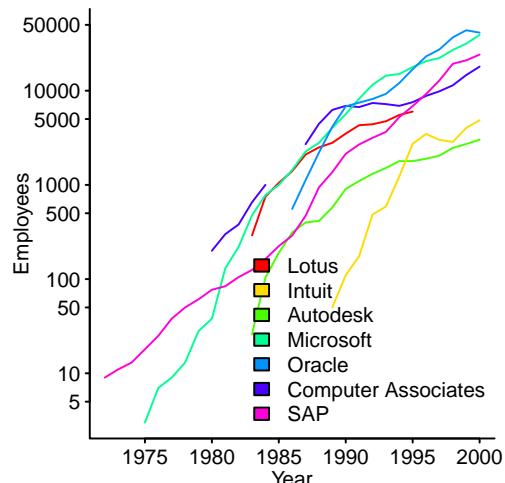


Figure 3.3: Number of people employed by major software companies. Data from Campbell-Kelly.<sup>246</sup> [code](#)

<sup>i</sup> The term is used generically, to refer to any software where the source code is freely available under a non-commercial license.

Some governments have recognized the importance of national software ecosystems,<sup>1179</sup> both in economic terms (e.g., industry investment in software systems<sup>284</sup> that keep them competitive), and as a means of self-determination (i.e., not having important infrastructure dependent on companies based in other countries); there is no shortage of recommendations<sup>1517</sup> for how to nurture IT-based businesses.

The software export figures given for a country can be skewed by a range of factors, and the headline figures may not include associated costs.<sup>674</sup>

There has been discussion around the efficiency gains achieved by using computers not showing up in national economic indicators. Red Queen... data perhaps...?

What percentage of their income do software companies spend on developing software? A study by Mulford and Misra<sup>1114</sup> of 100 companies in Standard Industry Classifications (SIC) 7371 and 7372<sup>ii</sup>, with revenues exceeding \$100 million during 2014–2015, found that total software development costs were around 13% of revenue; see Figure 3.4. Marketing, profit, etc...

+

## 3.2 Investment decisions

Creating software is an irreversible investment, i.e., incomplete software has little or no resale value, and even completed software may not have any resale value.

Is it worthwhile investing resources, to implement software that provides some desired functionality? In the case of a personal project, an individual may decide on the spur of the moment, to spend time implementing or modifying a program; for commercial projects a significant investment may be made analyzing the potential market, performing a detailed cost/benefit analysis, and weighing the various risk factors.

This sections outlines the major factors involved in making investment decisions, and some techniques that may be used. While a variety of sophisticated techniques are available, managers may choose to use the simpler ones<sup>1533</sup>...

The term *cost/benefit* applies when making a decision about whether to invest or not; while the term *cost-effectiveness* applies when a resource is available, and has to be used wisely, or when an objective has to be achieved as cheaply as possible.

- uncertainty...
- the value of money over time. A pound/dollar in the hand today is worth more than a pound/dollar in the hand tomorrow.
- Opportunity cost...
- risks... Investors require a greater return from a high risk investment, than from a low risk investment.

*Return on investment (ROI)* is defined as:

$$R_{est} = \frac{B_{est} - C_{est}}{C_{est}}$$

where:  $B_{est}$  is the estimated benefit, and  $C_{est}$  the estimated cost.

Both the cost, and the benefit estimates are likely to contain some amount of uncertainty, and the minimum and maximum ROI are given by:

$$R_{est} - \delta R = \frac{B_{est} - \delta B}{C_{est} + \delta C} - 1 \quad \text{and} \quad R_{est} + \delta R = \frac{B_{est} + \delta B}{C_{est} - \delta C} - 1$$

where:  $\delta$  is the uncertainty in the corresponding variable.

However, ROI uncertainty is unlikely to take an extreme value, and its expected value is given by.<sup>187</sup>

<sup>ii</sup> Computer programming services and Prepackaged software.

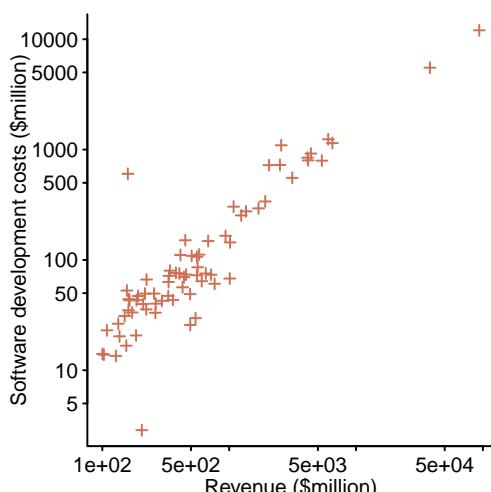


Figure 3.4: Company revenue (\$millions) against total software development costs. Data from Mulford et al.<sup>1114</sup> code

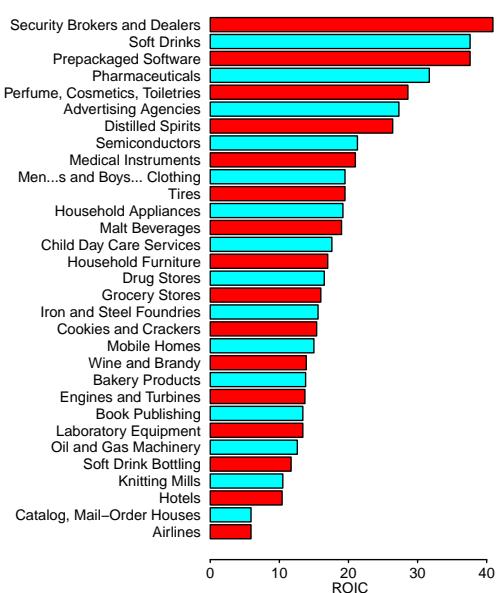


Figure 3.5: Average Return On Invested Capital of various U.S. industries between 1992-2006. Data from Porter.<sup>1266</sup> code

$$E[\delta R] \approx \frac{B_{est}}{C_{est}} \sqrt{\left(\frac{\delta B}{B_{est}}\right)^2 + \left(\frac{\delta C}{C_{est}}\right)^2}$$

When a resource is invested, the opportunity to use it to make an alternative investment, possibly with a greater return, is lost; this loss of opportunity is known as an *opportunity cost*. For instance, when deciding between paying for Amazon spot instances<sup>139</sup> at a rate similar to everybody else, and investing time trying to figure out an algorithm that makes it possible to successfully bid at much lower prices, the time spent figuring out the algorithm is an opportunity cost (i.e., the time spent is a lost opportunity for doing something else, which may have been more profitable).<sup>iii</sup>

Figure 3.6 shows the development cost of video games (where the cost was more than \$50million). The high risk of a market that requires a large upfront investment, to create a new product for an uncertain return, is offset by the possibility of a high return.

### 3.2.1 Discounting for time

A dollar today is worth more than a dollar tomorrow, because today's dollar can be invested and earn interest; by tomorrow, the amount could have increased in value (or at least not lost value, through inflation). The present value ( $PV$ ) of a future payoff,  $C$ , can be calculated from:

$$PV = \text{discount\_factor} \times C$$

where:  $\text{discount\_factor} < 1$ ; it is usually represented by:  $\text{discount\_factor} = (1+r)^{-1}$ , where:  $r$  is known as the *rate of return* (also known as the *discount rate*, or the *opportunity cost of capital*), and represents the size of the reward demanded by investors for accepting a delayed payment (it is often quoted over a period of a year).

The  $PV$  over  $n$  years (or whatever period  $r$  is expressed in) is given by:

$$PV = \frac{C}{(1+r)^n}$$

When comparing multiple options, expressing each of their costs in terms of present value enables them to be compared on an equal footing.

For example, consider the choice between spending \$250,000 purchasing a test tool, or the same amount on hiring testers; assuming the tool will make an immediate cost saving of \$500,000 (by automating various test procedures), while hiring testers will result in a saving of \$750,000 in two years time. Which is the more cost-effective investment (assuming a 10% discount rate)?

$$PV_{tool} = \frac{\$500,000}{(1+0.10)^0} = \$500,000 \quad \text{and} \quad PV_{testers} = \frac{\$750,000}{(1+0.10)^2} = \$619,835$$

Based on these calculations, hiring the testers is more cost-effective, i.e., it has the greater present value.

### 3.2.2 Taking risk into account

The calculation in the previous section assumed there was no risk of unplanned events. What if the tool did not perform as expected, what if some testers were not as productive as hoped? A more realistic calculation of present value needs to take into account the possibility that future payoffs are smaller than expected.

A risky future payoff is worth less than a certain future payoff, for the same amount invested and payoff. Risk can be factored into the discount rate, to create an *effective discount rate*:  $k = r + \theta$  (where:  $r$  is the risk-free rate, and  $\theta$  a premium that depends on the amount of risk). The formulae for present value becomes:

$$PV = \frac{C}{(1+k)^n}$$

When  $r$  and  $\theta$  can vary over time, we get:

<sup>iii</sup> There is also the risk that the result of successfully reverse engineering the pricing algorithm results in Amazon changing the algorithm.<sup>139</sup>

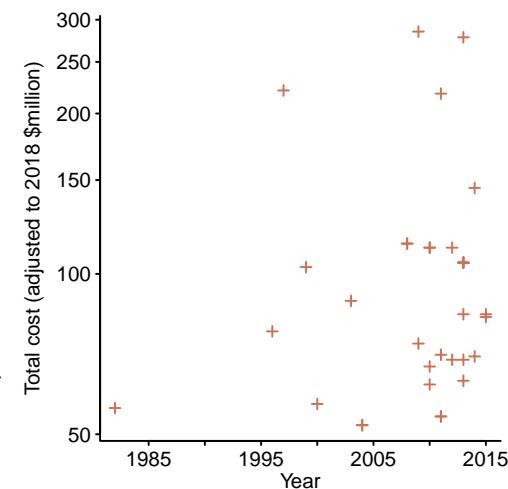


Figure 3.6: Development cost (adjusted to 2018 dollars) of computer video games, whose cost was more than \$50million. Data from Wikipedia.<sup>1663</sup> code

$$PV = \sum_{i=1}^t \frac{return_i}{(1+k_i)^i}$$

where:  $return_i$  is the return during period  $i$ .

Repeating the preceding example, assuming a 15% risk premium for the testers option, we get:

$$PV_{tool} = \frac{\$500,000}{(1+0.10)^0} = \$500,000 \quad \text{and} \quad PV_{testers} = \frac{\$750,000}{(1+0.10+0.15)^2} = \$480,000$$

Taking an estimated risk into account, suggests that buying the tool is the most cost-effective of the two options.

The previous analysis compares the two benefits, but not the cost of the investment that needs to be made to achieve the respective benefit. Comparing investment costs requires taking into account when the amounts were spent, to calculate the total cost terms of a present cost.

Purchasing the tool is a one time, up front, payment:

$$investment\_cost_{tool} = \$250,000$$

The cost of the testers approach is more complicated; let's assume it is dominated by monthly salary costs. If the testing cost is \$10,416.67 per month for 24 months, the total cost after two years, in today's terms, is (a 10% annual interest rate is approximately 0.8% per month):

$$investment\_cost_{testers} = \sum_{m=0}^{23} \frac{\$10,416.67}{(1+0.008)^m} = \$10,416.67 \left[ \frac{1 - (1 + 0.008)^{-22}}{1 - (1 + 0.008)^{-1}} \right] = \$211,042.90$$

Spending \$250,000 over two years is equivalent to spending \$211,042.90 today. Investing \$211,042.90 at 10% today, would provide a fund that supports spending \$10,416.67 per month for 24 months.

*Net Present Value* (NPV) is defined as:  $NPV = PV - investment\_cost$

Plugging in the calculated values gives:

$$NPV_{tool} = \$500,000 - \$250,000 = \$250,000$$

$$NPV_{testers} = \$480,000 - \$211,042.90 = \$268,957.10$$

Based on NPV, hiring testers is the more cost-effective option.

Alternatives to NPV, their advantages and disadvantages, are discussed by Brealey<sup>205</sup> and Raffo.<sup>1305</sup> One commonly encountered rule, in rapidly changing environments, is the pay-back rule, which requires that the investment costs of a project be recovered within a specified period; the *payback period* is the amount of time needed to recover investment costs (a shorter payback period being preferred to a longer one).

Accurate estimates for the NPV of different options, requires accurate estimates for the discount rate, and the impact of risk. The discount rate represents the risk-free element, and the closest thing to a risk-free investment is government bonds and securities (information on these rates is freely available). Governments face something of a circularity problem in how they calculate the discount rate for their own investments. The US government discusses these issues in its "Guidelines and Discount Rates for Benefit-Cost Analysis of Federal Programs",<sup>1656</sup> and at the time of this writing the revised Appendix C specified rates, varied between 0.9% over a three-year period and 2.7% over 30 years. Commercial companies invariably have to pay higher rates of interest than the US Government.

### 3.2.3 Incremental investments and returns

Sometimes investments and returns are incremental, occurring at multiple points in time, over an extended period.

The following example is based on the idea that it is possible to make an investment, when writing or maintaining code, that reduces subsequent maintenance costs, i.e., produces a return on the investment. At a minimum, any investment made to reduce later maintenance costs must be recouped; this minimum case has an ROI of 0%.

Let  $d$  be the original development cost,  $m$  the base maintenance cost during time period  $t$ , and  $r$  the interest rate; to keep things simple assume that  $m$  is the same for every period of maintenance; the NPV for the payments over  $t$  periods is:

$$\text{Total\_cost} = d + \sum_{k=1}^t \frac{m}{(1+r)^k} = d + m \left[ \frac{1 - (1+r)^{-(t+1)}}{1 - (1+r)^{-1}} - 1 \right] \approx d + m \times t [1 - 0.5 \times (t+1)r]$$

with the approximation applying when  $r \times t$  is small.

If an investment is made for all implementation work (i.e., development cost is:  $(1+i)d$ ), in expectation of achieving a reduction in maintenance costs during each period (by  $1-b$ ), then:

$$\text{Total\_cost}_{\text{invest}} = (1+i)d + m \times (1+i)(1-b) \times t [1 - 0.5 \times (t+1)r]$$

For this investment to be worthwhile:  $\text{Total\_cost}_{\text{invest}} \leq \text{Total\_cost}$ , giving:

$$(1+i)d + m \times (1+i)(1-b) \times T < d + m \times T$$

where:  $T = t [1 - 0.5 \times (t+1)r]$ , which simplifies to give the following lower bound for the benefit/investment ratio,  $\frac{b}{i}$ :

$$1 + \frac{d}{mT} < \frac{b}{i} + b$$

In practice many systems have a short lifetime. What value must the ratio  $\frac{b}{i}$  have, for an investment to break even, after taking into account system survival rate (i.e., the possibility that there is no future system to maintain)?

If  $s$  is the probability the system survives a maintenance period, the total system cost is:

$$\text{Total\_cost} = d + \sum_{k=1}^t \frac{m \times s^k}{(1+r)^k} = d + m \frac{s(1-s^t)}{1-s}$$

where:  $S = \frac{s}{1+r}$ . The minimal requirement is now:

$$1 + \frac{d}{m} \frac{1-S}{S(1-S^t)} < \frac{b}{i} + b$$

The development/maintenance break-even ratio depends on the regular maintenance cost, multiplied by a factor that depends on the system survival rate (not the approximate total maintenance cost).

Fitting regression models to the system lifespan data in [?], finds (for an annual maintenance period):  $s_{\text{mainframe}} = 0.87$  and  $s_{\text{Google}} = 0.79$ ; information on possible development/maintenance ratios is only available for mainframe software (see Figure 4.43), and the annual mean value is:  $\frac{d}{m} = 4.9$ .

Figure 3.7 shows the multiple of any additional investment, made during development, that needs to be recouped during subsequent maintenance work, to break even for a range of payback periods (when application survival rate is that experienced by Google applications, or 1990s Japanese mainframe software; the initial development/annual maintenance ratios are 5, 10 and 20); the interest rate used is 5%.

This analysis only considers systems that have been delivered and deployed; projects are sometimes cancelled before reaching this stage, and including these in the analysis would increase the benefit/investment break-even ratio that needs to be achieved.

While a large percentage<sup>431</sup> of a program's features may never be used, those features that will not be used is unknown at the time of implementation...

Figure 7.21 shows that most files are only ever edited by one person...

### 3.2.4 Investment under uncertainty

The future uncertain, and the analysis in the previous sections assumed fixed values, for future rates of interest and risk. A more realistic analysis would take into account future uncertainty.<sup>408,735</sup>

An investment might be split into random and non-random components, e.g., Brownian motion with drift. The following equation is used extensively to model the uncertainty involved in investment decisions<sup>408</sup> (it is a stochastic differential equation for the change in the quantity of interest,  $x$ ):

$$dx = a(x,t)dt + b(x,t)dz$$

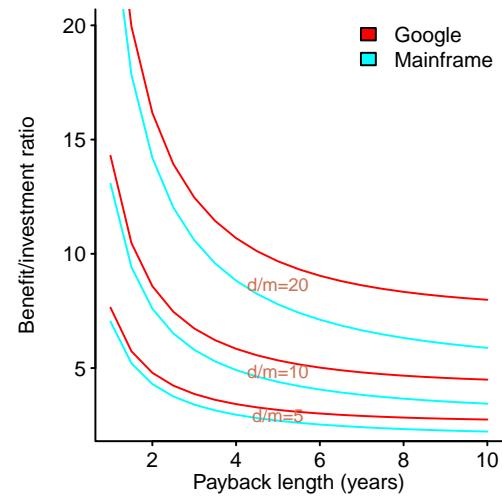


Figure 3.7: Return/investment ratio needed to break-even, for Google and Mainframe application survival rate, having development/annual maintenance ratios of 5, 10 and 20; against payback period in years. Data from: mainframe Tamai,<sup>1534</sup> Google SaaS Ogden.<sup>1181</sup> code

This equation contains a drift term (given by the function  $a$ , which involves  $x$  and time) over an increment of time,  $dt$ , plus an increment of a Wiener process,<sup>iv</sup>  $dz$  (the random component; also known as a Brownian process), and the function,  $b$ , involves  $x$  and time.

The simplest form of this equation is:  $dx = \alpha dt + \sigma dz$ , i.e., a constant drift rate,  $\alpha$ , plus a random component having variance  $\sigma$ . Figure 3.8 illustrates this drift-diffusion process; with the grey line showing the slope of the drift component, and green lines showing possible paths followed when random increments are added to the drift (red lines bound possible paths, for the value of  $\sigma$  used).

This equation can be solved to find the standard deviation in the value of  $x$ , it is  $\sigma\sqrt{T}$ , where:  $T$  is elapsed time...

The analysis of the maintenance investment cost/benefit, in the previous section, assumed a fixed amount of maintenance in each interval. In practice, the amount of maintenance is likely to grow to a maximum commercially supportable value, and then fluctuate about this value over time, i.e., it becomes a mean-reverting process. The *Ornstein-Uhlenbeck process* (also known as the *Vasicek process*) is the simplest mean-reverting process, and its corresponding equation is:

$$dx = \eta(\hat{x} - x)dt + \sigma dz$$

where:  $\eta$  is the speed of reversion, and  $\hat{x}$  is the mean value.

This equation can be solved to give the expected value of  $x$  at future time  $t$ , given its current value  $x_0$ :  $E[x_t] = \hat{x} + (x_0 - \hat{x})e^{-\eta t}$ ; its variance is:  $\frac{\sigma^2}{2\eta}(1 - e^{-2\eta t})$ . Section 11.10.10 discusses techniques for obtaining values for  $\sigma$  and  $\eta$ .

Figure 3.9 shows ten example paths (in green) of an Ornstein-Uhlenbeck process, each starting at zero, and growing to fluctuate around the process mean (red line is the expected value, blue lines are at one standard deviation).

Uncertainty in the investment cost and/or the likely benefit returned from a project, means that even under relatively mild conditions, a return of twice as much as the investment is considered to be the break-even condition.<sup>408, 1250</sup>

Building a software system involves making sequential investments, and there is a maximum construction rate; investment problems having these characteristics occur in factory construction<sup>1009</sup> ...

Obtaining insight about a possible investment, by setting up and solving a stochastic differential equation<sup>1397</sup> can be very difficult, or impractical. A variety of numerical methods are available for analyzing investment problems based on a stochastic approach, see Section 11.10.10.

Current market conditions also need to be taken into account, for instance: how likely is it that other companies will bring out competing products? Will demand for the application still be there once development is complete?

### 3.2.5 Real options

It may be possible to delay making an investment until circumstances are more favorable.<sup>408</sup> For instance, when new functionality has to be implemented as soon as possible, a decision may be made to delay investing the resources needed to ensure the new code conforms to project guidelines; there is always the option to invest the necessary resources later. By using Net Present Value, management is taking a passive approach to their investment strategy; a fixed calculation is made, and subsequent decisions are based on the result.

In financial markets, a *call option* is a contract between two parties (a buyer and a seller), where the buyer pays the seller for the right (but not the obligation) to buy (from the seller) an asset, at an agreed price on an agreed date (the initial amount paid is known as the *premium*, the agreed price the *strike price*, and the date the *expiry* or *maturity* date). When the option can only be exercised on the agreed date, the contract is known as a *European option*, when the option can be exercised at any time up to the agreed date, it is known as a *American option*.

<sup>iv</sup> A Wiener process involves random incremental changes, with changes being independent of each other, and the size of the change having a Normal distribution.

A *put option* is a contract involving the right (but not the obligation) to sell.

The term *real options* (or *real options valuation ROV*, or *real options analysis*) is applied to the analysis of decisions made by managers in industry that have option-like characteristics. In this environment many items are not traded like conventional financial options; another notable difference, is that those involved in the management of the asset, may have an influence on the value of the option (e.g., they have a say in the execution of a project).

In financial markets, volatility information can be obtained from an assets past history. In industry, managers may have information on previous projects carried out within the company, but otherwise have to rely on their own judgment to estimate uncertainty. Call and put options are established practice in financial markets. In industry, managers have to create or discover possible options; they need an entrepreneurial frame of mind.

ROV requires active management, continuously ready to respond to changing circumstances. It is an approach that is most applicable when uncertainty is high, and managers have the flexibility needed to make the required changes.

Option to expand, contract, abandon...

sequencing of options: observing the outcome of the first project, using this new information to resolve some of the uncertainty relating to the overall project...

Binomial model...

Implementing functionality using the minimum of investment, with the intent of investing more later, has the form of an American call option.<sup>v</sup> The call option might not be exercised, e.g., if the implemented functionality became unnecessary.

The Black-Scholes equation provides a solution to the optimal pricing of European options (e.g., the value of the premium, strike price, and maturity date). Some researchers have applied this equation to the options associated with developing software; this is a mistake.<sup>490</sup> The derivation of the Black-Scholes equation involves several preconditions, which often apply to financial risks, but don't apply to software development, including:

- liquidity is required, to enable the composition of a portfolio of assets to be changed, at will, by investing or divesting (i.e., buying or selling). Software production does not create liquid assets, the production is a sunk cost; once an investment has been made in writing code, it is rarely possible to immediately divest a percentage of the investment in this code (a further investment in writing new code may make little business sense).
- detailed historical information on the performance of the item being traded is an essential input to portfolio risk calculations. Historical data is rarely available on one, let alone all, of the major performance factors involved in software development; Chapter 5 discusses the distribution of various development activities over the lifetime of a development project.

A cost/benefit analysis of decisions involving whether to fix a known fault now, has to take into account the probability that it will never be experienced by a customer (see Figure 11.76), and the possibility of reputational damage if it results in a major loss for important customers.

### 3.3 Capturing cognitive output

Companies, where the cognitariate are responsible for a significant percentage of profit, are becoming social factories,<sup>259</sup> involving themselves in employees' lives to reduce the external frictions. The intent is to maximise the capture of employee cognitive output; free meals and laundry service are not perks, they are a means of bringing employees together to learn and share ideas, by reducing opportunities to mix in non-employee social circles (and creating employee life norms that limit possible alternative employers, i.e., a means of reducing knowledge loss and spillover).

Companies that continue to be based around work-life separation also use software systems, and offer software related jobs. In these companies the focus may be on the cognitive efforts of people working on non-software activities, with those working in software related activities having to fit in with the existing workplace norms of other industries.

---

<sup>v</sup> A term in common usage is *technical debt*; this is incorrect, there is no debt.

Some developers are more strongly motivated by enjoyment of doing what they do, than the money they receive for doing it. This lifestyle choice relies on the willingness of companies to tolerate workers who are unwilling to do work they don't enjoy, or alternating between high paying work that is not enjoyed, and working for pleasure. Freelancing in tasks such as trying to earn bug bounties is only profitable for a few people (see Figure 4.39).

A study by Bao, Xing, Xia, Lo and Li<sup>101</sup> investigated software development staff turnover in two large companies... [projects/WhoWillLeaveCompany.R](#)

Choosing who to employ...?

### 3.3.1 Intellectual property

Governments have created two legal structures that might be used by individuals, or organizations, to claim property right over particular source code, or ways of doing things (e.g., business processes): patents and copyright.

A patent gives its owner rights to exclude others from making or selling an item that makes use of the claims made in the patent (for a term of 20 years from the date of filing; variations apply). In the U.S. software was not patentable until 1995, and since then the number of software patents has continued to grow, with 109,281 granted in 2014<sup>1163</sup> (36% of all utility patents).?

?

A license is a legal document, chosen or created by the copyright owner of software, whose intended purpose is to control the use, distribution, and creation of derivative works of the software. Licensing is an integral component of the customer relationship.

Licensing is a source of ecological pressure (e.g., applications available under a commercial license, may find it difficult to remain viable in ecosystems containing similar applications, available under an open source license); the ideology associated with some kinds of licenses may be strong enough for people to create license-based ecosystems (e.g., Debian distributions only contain software whose source is licensed under a Free Software license). The restrictions imposed by a license, on the usage of a software system, may be sufficiently at odds with the ideology of some developers, that they decide to invest their time in creating a new implementation of the functionality under a different license.

People have been making source code available at no cost, for others to use, since software was first created;<sup>54</sup> however, the formal licensing of such software is relatively new.<sup>1347</sup> Open source licenses have evolved in response to user needs, court decisions, and the growth of new product ecosystems; there has been a proliferation of different, and sometimes incompatible, open source licenses.<sup>1092</sup> Factors driving the evolution of licensing conditions, and combinations of conditions, include: evolution of the legal landscape (e.g., issues around what constitutes distribution and derivative works<sup>637</sup>), commercial needs for a less restrictive license (e.g., the 3-clauses and 2-clauses variants of the original, more restrictive BSD license, now known as 4-clauses BSD), ideological demands for a more restrictive license (e.g., the creation of GPL<sup>890</sup> version 3 added wording to address issues such as hardware locks and digital rights management, that were not addressed in GPL version 2), the desire to interoperate with software made available under an incompatible, but widely used, license (often the GPL),<sup>561</sup> or the desire to operate a business supporting a particular open source software system (e.g., dual licensing<sup>1584</sup>).

License selection, by commercial vendors, is driven by the desire to maximise the return on investment. Vendor licensing choice can vary from a license that ties software to a designated computer (with the expectation that the customer will later buy licenses for other computers), to a very permissive open source license<sup>vi</sup> (whose intent is to nullify a market entry-point for potential competitors).<sup>248</sup>

The majority of source code files do not contain an explicit license.<sup>1601</sup> Developers who do specify a license, may be driven by ideology, or the desire to fit in with the license choices made by others in their social network.<sup>1449</sup> One study<sup>27</sup> found that developers were able to correctly answer questions involving individual licenses (as-in, the answer agreed with a

<sup>vi</sup> The term *permissive* is used to denote the extent to which an open source license, consistent with a particular ideological viewpoint, allows such licensed software to be used, modified and operated in conjunction with, other software licensed under a license consistent with another ideological viewpoint.

legal expert), but struggled with questions that required integrating conditions contained in multiple licenses.

Commercial vendors may invest in a new implementation of the functionality provided by an existing software system, because they want the licensing fee income, or because the existing software has an open source license that does not suit their needs. For instance, Apple have been a longtime major funder of the LLVM compiler project, an alternative to GCC (one is licensed under the GPL, the other under the University of Illinois/NCSA open source license; the latter is a more permissive license).

Details of any applicable license(s) may appear within individual files, and/or in a license file within the top-level directory of the source code. In the same way that software can be changed, the license under which software is made available can change.<sup>1616</sup> Any changes to the licensing of a software system become visible to users when a new release is made.

When some files used to build a software system contain a license, different licenses may appear in different files; the different licenses may specify conditions that are incompatible, with each other. For instance, open source licenses might be classified as restrictive (i.e., if a modified version of the software is distributed, the derived version must be licensed under the same terms), or permissive (i.e., derived works may incorporate software licensed under different terms).

A study by Vendome, Linares-Vásquez, Bavota, Di Penta, German and Poshyvanyk<sup>1601</sup> investigated license usage in the files of 16,221 Java projects (the later study also included projects written in other languages). Nearly all files did not include a license, and when a license was present it was rarely changed (the most common change, by an order of magnitude, was to/from a license being present).

Figure 3.11 shows the cumulative percentage of licenses present in files, over time, in the 10% of projects having the most commits. The lines show 16 of the 80 different licenses appearing in files, with the other 64 licenses appearing in less than 1,000 files. The downward trend, for some licenses, is caused by the by an increasing project growth rate, with most files not including any license (upper line in plot).

A study by German, Manabe and Inoue<sup>562</sup> investigated the use of licenses in the source code of programs in the Debian distribution (version 5.0.2). They found that 68.5% of files contained some form of license statement (the median size of the license was 1,005 bytes). The median size of all files was 4,633 bytes; the median size of files without a license 2,137 bytes, and the files with a license 5,488.

Programs and libraries may have dependencies on packages made available under licenses, which specify conditions that are incompatible, with each other.

A study by Meloca, Pinto, Baiser, Mattos, Polato, Wiese, and German<sup>1063</sup> investigated licensing files appearing in all packages from the NPM (510,964 packages), RubyGems (135,481 packages), and CRAN (11,366 packages) networks (from the launch of the package site to October 2017).

Figure 3.12 shows the number of package releases (some packages were updated, and an updated version released; there were 4,367,440 releases), containing a given number of licenses (a package must contain a license to appear on CRAN).

A study of 857 mobile apps,<sup>1098</sup> from the F-droid market, found license inconsistency potentially as high as 85%...

When licensing conditions are not followed, what action, if any, should the copyright holders of the software take?

The licensee has the option of instigating legal proceedings to protect their property. Those seeking to build market share may not take any action against some license violations.

The response of the open source community... If the copyright holders signal that they plan to take legal action, a functionally equivalent...

Some companies and individuals have sought to obtain licensing fees from users of some software systems, made available under an open source license. If the licensing fee sought is small enough, it may be cheaper for larger companies to pay, rather than legally contest the claims.<sup>1652</sup>

While many people do not read end-user license agreements (EULAs),<sup>1021</sup> ...

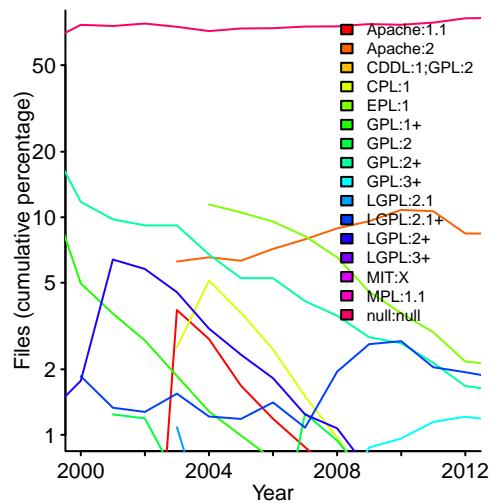


Figure 3.11: Cumulative percentage of files, from the top 10% largest Java projects, containing a given license (upper line is no license). Data from Vendome et al.<sup>1601</sup> code

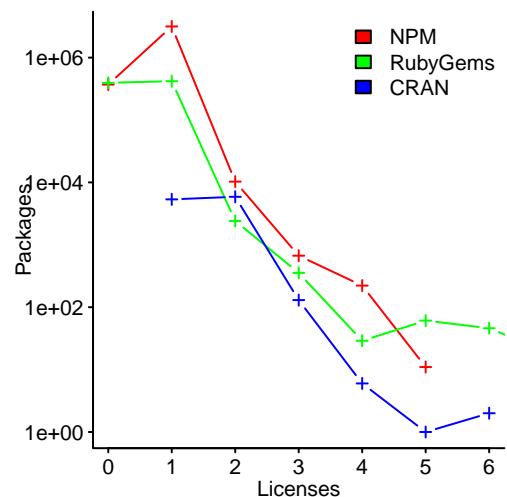


Figure 3.12: Number of releases of packages containing a given number of licenses (a package has to contain a license to appear on CRAN). Data from Meloca et al.<sup>1063</sup> code

A license is interpreted within the framework of the laws of the country in which it is being used. For instance, the Uniform Commercial Code (UCC) is a series of laws, intended to harmonise commercial transactions across the US; conflicts between requirements specified in the UCC, and a software license may have been resolved by existing case law, or may be resolved by a new court case.<sup>1047</sup> Competing against free is very difficult. The Free Software Foundation were the (successful) defendants in a case where the plaintiff argued that the GPL violated the Sherman Act, e.g., “the restraint of trade by way of a licensing scheme to fix the prices of computer software products”.<sup>1555</sup>

A country’s courts have the final say in how the wording contained in software licenses is interpreted. The uncertainty over the enforceability<sup>589,895</sup> of conditions contained in open source licenses is slowly being clarified, as more cases are being litigated, in various jurisdictions.<sup>968,1567</sup>

In litigation involving commercial licenses, the discovery that one party is making use of open source software, without complying with the terms of its open source license, provides a bargaining chip for the other party,<sup>1453</sup> i.e., if the non-compliance became public, the cost of compliance, with the open source license, may be greater than the cost of the commercial licensing disagreement.

The Open Source Initiative (OSI) is a non-profit organization, that has defined of what constitutes open source, and maintains a list of licenses that meet this definition. OSI has become the major brand, and largest organization operating in this area; licenses submitted, to become OSI-approved, have to pass a legal and community wide review.

Table 3.1 shows the percentage of all NPM package releases using a given license, or lack thereof (column names), that migrated (or not) to another license (row names), from the launch of the NPM network in 2009, until October 2017. During this period 85% of all package releases contained an OSI license; see [economics/msr2018b\\_evol.R](#).

To/From	OSI	Incomplete	non-OSI	Missing	Other	Copyright
OSI	99.7	6.8	7.7	2.6	4.8	2.8
Incomplete	0.12	92.0	0.1	0.09	3.3	3.0
non-OSI	0.08	0.07	91.1	0.07	0.95	0.81
Missing	0.08	0.22	0.65	97.2	0.49	0.26
Other	0.02	0.54	0.28	0.02	88.1	3.7
Copyright	0.00	0.31	0.2	0.01	2.3	90.0

Table 3.1: Percentage migration of licenses used by all NPM package releases (between 2009 and 2017), from license listed in column names, to license in row names. Data from Meloca et al.<sup>1063</sup>

Figure 3.13 shows the survival curve for the 107 OSI licenses that have been listed on the OSI website since August 2000; at the start of 2019, 81 licenses were listed on the OSI website.

A software system may be dependent on particular data, for correct or optimal operation. The licensing of data associated with software, or otherwise, can be an important issue itself.<sup>672</sup>

Open source ideology is being applied to other goods, in particular the hardware on which software runs.<sup>181</sup>

### 3.3.2 Bumbling through life

The expectation of career progression comes from an era when many industries were stable, for long enough, for career paths to become established. In rapidly changing working environments, the concept of career progression may not apply. Some software companies have established career progression ladders for their employees;<sup>1479</sup> Gruber<sup>636</sup> discusses the issues for IT careers, in detail.

Companies have an interest in offering employees a variety of career options, it is a means of retaining the knowledge acquired by employees based on their experience of the business. Employees may decide that they prefer to continue focusing on technical issues, rather than people-issues, or may embrace a move to management. Simulations<sup>1255</sup> have found some truth to the Peter principle: “Every new member in a hierarchical organization climbs the hierarchy until they reach their level of maximum incompetence.”

In fast changing knowledge-based industries, an individuals' existing skills can quickly become obsolete. Options available to those with skills which are likely to become obsolete include, include having a job that looks like it will continue to require the learned skills for a many years, and learning appropriate new skills.

Employee turnover... investing to make it cheaper for new hires to learn existing code, compared to the cost of forgetting by the person who wrote it... people who prefer to write software, people who are happy to maintain software, people who want to manage... [Turnover Rates/...](#)

also see Figure 3.23

Mean time working at a company...

Human capital theory suggests there is a strong connection between a person's salary and time spent on the job at a company, i.e., the training and experience gained over time increases the likelihood that a person could get a higher paying job elsewhere; it is in a company's interest to increase employee pay over time<sup>127</sup> (one study<sup>1456</sup> of 2,251 IT professionals, in Singapore, found a linear increase in salary over time).

Regional pay differences... Estimating sizes of communities. [ecosystems/MSA\\_M2016\\_CM.R](#)  
bls occupational Employment Statistics [oes/](#), [O\\_Net](#)

A study by Joseph, Boh, Ang and Slaughter<sup>809</sup> analyzed BLS data to find the sequence of job held by people working in IT... extract data...

Implementing new software is seen, by some developers, as being much more interesting, and rewarding, than maintaining existing software.<sup>7</sup> It is common for the members of the original software to move on to other projects, once the one they are working on is initially completed.<sup>7</sup> A study by Couger and Colter<sup>337</sup> investigated approaches to motivating developers working on maintenance activities; they identified the following factors:

- the motivating potential of the job: based on skill variety required, the degree to which the job requires completion as a whole (task identity), the impact of the job on others (task significance), degree of freedom in scheduling and performing the job, and feedback from the job (used to calculate a *Motivating Potential Score*, MPS),
- a person's need for personal accomplishment, to be stimulated and challenged; the term *growth need strength* (GNS) was used to describe this need.

The research provided support for the claim that MPS and GNS could be measured and that jobs could be tailored, to some degree, to people. Management's role was to organize the work that needed to be done to balance the MPS of jobs against the GNS of the staff available.

The Occupational Employment Statistics published by the BLS in the US and the Labour Force Survey published by the ONS in the UK are country averages over everybody who files a tax return... [oes/](#) [salary/](#)

In some US states, government employee salaries are considered to be public information, e.g., California (see [ecosystems/transparentcalifornia.R](#)); a few companies publish employee salaries, so called *Open salaries*; the information may be available to company employees only (e.g., ...), or it may be public (e.g., Buffer<sup>549</sup>).

If the male/female cognitive ability distribution seen in Figure 2.5 carries over to software competencies, then those seeking to attract more women into software engineering, and engineering in general, should be targeting the more populous middle competence band, and not the high-fliers. The mass market for those seeking to promote female equality is incompetence; a company cannot be considered to be gender neutral until incompetent women are equally likely to be offered a job as incompetent men.

Gender differences exist in the pay of IT occupations<sup>1151</sup>...

?

### 3.3.3 Expertise

To become a performance expert, a person needs motivation, time, economic resources, an established body of knowledge to learn from, and teachers to guide; while learning, performance feedback is required.

An established body of knowledge to learn from requires that a problem domain to have existed in a stable state for long enough for a proven body of knowledge to become established. The availability of teachers requires the domain be sufficiently stable that most of what potential teachers have learned is still applicable to students; if the people with the knowledge and skills are to be motivated to teach, they need to be paid enough to make a living.

The practice of software development is a few generations old, and has experienced a steady stream of substantial changes; substantial change is written about as-if it is the norm. Anybody who invests in many years of deliberate practice on a specific technology may find there are few customers for the knowledge and skill acquired, i.e., are willing to pay a higher rate to do the job, than that paid to somebody with a lot less expertise...

Software developers are not professional programmers, any more than they are professional typists; reading and writing software is one of the skills required to build a software system. Effort also needs to be invested in acquiring application domain knowledge and skills, for the markets targeted by the software system.

What level of software development related expertise is worth acquiring in a rapidly changing ecosystem? The level of skill required to obtain a job involving software development is judged relative to those who apply for the job... In an expanding market employers may have to make do with whatever they can get... Those with higher skill levels may have the luxury of being able chose the work that interests them...

Once a good enough level of programming proficiency is reached, if the application domain changes more slowly than the software environment, learning more about the application domain may provide a greater ROI, compared to improving programming proficiency (because the acquired application knowledge/skills have a longer usable lifetime)...

People often learn a skill for some purpose (e.g., chess as a social activity, programming because it provides pleasure or is required to get a job done), without aiming to achieve expert performance. Once a certain level of proficiency is achieved, such people stop trying to learn, and concentrate on using what they have learned; in work, and sport, a distinction is made between training for, and performing the activity. During everyday work, the goal is to produce a product, or to provide a service. In these situations people need to use well-established methods, to be certain of success, not try new ideas (which potentially lead to a dead-end, or to failure). Time spent on this kind of practice does not lead to any significant improvement in expertise, although it may lead to becoming very fluent in performing a particular subset of skills; computer users have been found to have distinct command usage habits.<sup>1390</sup>

Higher education once served as a signalling system,<sup>1475</sup> used by employers looking to recruit people at the start of their professional careers (i.e., high cognitive firepower was once required to gain a university degree). However, some governments' policy of encouraging a significant percentage of their citizens to study for a higher education degree, diluted university qualifications to being an indication of not below average IQ. By taking an active interest in the employability of graduates with a degree in a STEM related subject,<sup>1409</sup> governments are suffering from cargo-cult syndrome. Knowledge-based businesses want employees who can walk-the-talk, not drones who can hum a few tunes.

### 3.4 Group dynamics

People may voluntarily cooperate with others to complete a task that is beyond the capacity of an individual, for the benefit of all those involved; in a commercial environment people may not have much say over the group they are assigned to.

The effectiveness of a group is dependent on cooperation between its members ... Trust is an essential component of economic activity.<sup>364</sup>

Some groups may offer non-task specific benefits for individuals working within the group, e.g., social status and the opportunity for social learning.

Conflicts of interest can arise between the aims of the group and those of individual members, and those outside the group who pay for its services. Individuals may behave different when working on their own, compared to when working as a member of a group (e.g., social loafing); cultural issues (e.g., individualism vs. collectivism) are also a factor.<sup>426</sup>

??

When a group of people work together, a shared collection of views, beliefs and rituals (ways of doing things) is evolved,<sup>712</sup> i.e., a culture. Culture improves adaptability. Culture is common in animals, but cultural evolution<sup>1071</sup> is rare; perhaps limited to humans, song birds and chimpanzees.<sup>193</sup>

People and existing code are the carriers of software cultures.

While overconfidence at the individual level decreases the fitness of the entrepreneur (who does not follow the herd, but tries something new), the presence of entrepreneurs in a group increases group level fitness (by providing information about the performance of new ideas).<sup>151</sup>

Ethnographic studies of engineering culture have investigated a large high-tech company in the mid-1980s,<sup>896</sup> and an internet startup that had just IPO'ed.<sup>1349</sup>

### 3.4.1 Social status

The evidence<sup>41</sup> points to the desire for social status being a fundamental human motive. Higher status (or reputation) provides greater access to desirable things, e.g., jobs.<sup>1106</sup> To be an effective signalling system, social status has to be costly to obtain.<sup>682</sup>

Social animals pay more attention to group members having a higher social status (however that might be measured). People may seek out and pay deference to a highly skilled individual in exchange for learning access.

Social status and/or recognition is a currency that a group can bestow on members whose activities are thought to benefit the group. The failure of an academic community to treat the production of research related software as worthy of the appropriate academic recognition<sup>1118</sup> may slow the rate of progress; academic recognition is not always the primary motive for the creation of research software.<sup>728</sup>

When attention resources are constrained, people may choose to preferentially invest attention on high-status individuals.

A study by Simcoe and Waguespack<sup>1442</sup> investigated the impact of status on the volume of email discussion on proposals submitted to the Internet Engineering Task Force (IETF). A proposal submitted by an IETF working group had a much higher chance of becoming a published RFC, compared to one by a group of individuals (i.e., 43% vs. 7%). The IETF list server distributed proposals with the authors names, except during periods of peak load, when the author list was shortened (by substituting one generic author, "et al"). The, essentially random, shortening of the proposal author list created a nature experiment; the identity of the authors was available for some proposals, but not others. Proposals unlikely to become a published RFC (i.e., those submitted by individuals), would be expected to receive less attention, unless the list of authors included a high status individual (a working group chairman was assumed to be a high status individual).

An analysis of the number of email posts involving the 3,129 proposals submitted by individuals (mean 3.4, sd 7.1), found: 1) when a working group chair appeared on the author list, the weighted number of responses increased by 0.8, and 2) when a working group chair name was one of the authors but had been replaced by "et al", the weighted number of responses fell by 1.7; see [economics/EtAIData.R](#).

Figure 3.14 shows the number of proposals receiving a given number of mentions in IETF email discussions, with lines showing fitted regression models; colors denote whether the author list included a working group chairman and whether this information was visible to those receiving the email.

Fashion as a prestige signalling system...

Size of crowd looking up increases with size of stimulus crowd<sup>1078</sup>...

### 3.4.2 Social learning

Learning by observing others, *social learning*, enables animals to avoid the potentially high cost of *individual learning*.<sup>50,723</sup> A population's average fitness increases when its members are capable of deciding, which of two learning strategies, social learning or individual learning, is likely to be the most cost effective<sup>192</sup> (i.e., the cost of individual learning can be invested where its benefit is likely to be maximized).

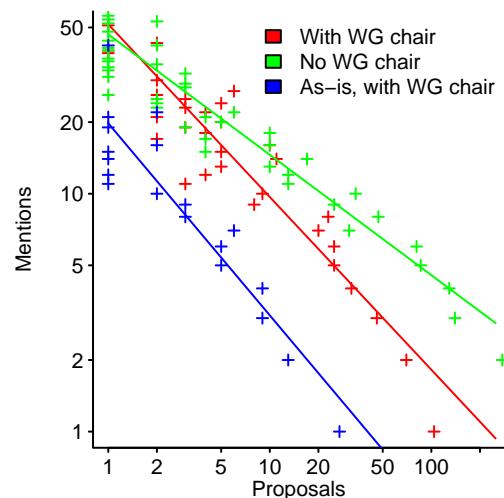


Figure 3.14: Number of proposals receiving a given number of mentions in emails, and lines showing fitted regression models. Data from Simcoe et al.<sup>1442</sup> [code](#)

Prestige-biased transmission occurs when people select the person with the highest prestige, as being the most likely to possess adaptive information<sup>684</sup> for a learner to imitate.

Duplicating the behavior of others is not learning, it is a form of *social transmission*. Following leaders generates a pressure to conform, as does the desire to fit in with a group, known as *conformist transmission*...

In England, milk was once left in open bottles on customer doorsteps; various species of birds were known to drink some of this milk. When bottles were sealed (these days with aluminium foil), some birds learned to peck open milk bottle tops, with the blue tit being the primary scavenger.<sup>501vii</sup> The evidence suggests those bird species that forage in flocks, have the ability to learn both socially and individually, while species that are territorial (i.e., chase away other members of their species), primarily use individual learning.<sup>938</sup>

Social learning is a skill, where 2.5-year-old children significantly outperform adult chimpanzees and orangutans, our closest primate relatives<sup>690</sup> (performance on other cognitive tasks is broadly comparable).

Just using social learning is only a viable strategy in an environment that is stable between generations of learners; if the environment is likely to change between generations, copying runs the risk of learning skills that are no longer effective. Analysis of a basic model<sup>1043</sup> shows that for a purely social learning strategy to spread in a population of individual learners, the probability of environmental change in any given generation,  $u$ , and learning costs,  $b$  is the cost of individual learning and  $c$  the reduction in the cost of learning achieved by social learning, the following relation must be true:  $u > \frac{c}{b}$ .

A study by Centola, Becker, Brackbill and Baronchelli<sup>261</sup> investigated the relative size of subgroups acting within a group, needed to change a social convention previously established by group members. Groups containing between 18 and 30 people worked in pairs, with pairs changing after every interaction, performing a naming task; once consistent naming conventions had become established within the group, a subgroup were instructed to use an alternative naming convention. The results found that a committed subgroup containing at least 25% of the group were able to cause the group to switch, to using the subgroup's naming conventions. The rate of change depended on group size and relative size of the subgroup; see [economics/Centola-Becker.R](#)

Discoveries and inventions are often made by individuals, and it might be expected that larger populations will contain more tools and artefacts, and have a more complex cultural repertoire than smaller populations.<sup>942</sup> The evidence is mixed, with population size having a positive correlation with tool complexity in some cases.<sup>314</sup>

Useful new knowledge is not always uniformly diffused out into the world, to be used by others; a constantly changing environment introduces uncertainty<sup>1282</sup> (i.e., noise is added to the knowledge signal), and influential figures may suppress use of the ideas (e.g., some physicists suppressed the use of Feynman diagrams<sup>155</sup>).

Conformist transmission is the name given to the hypothesis that individuals possess a propensity to preferentially adopt the cultural traits that are most frequent in the population. Under conformist transmission, the frequency of a trait among the individuals within the population provides information about the trait's adaptiveness. This psychological bias makes individuals more likely to adopt the more common traits than they would under unbiased cultural transmission. Unbiased transmission may be conceptualized in several ways. For example, if an individual copies a randomly selected individual from the population, then the transmission is unbiased. If individuals copy their parents or just their mother, then transmission also is unbiased.

The rate of change in the popularity lists of baby names, dog breeds and pop music has been fitted<sup>144</sup> by a model where most of the population,  $N$ , randomly copy an item on the list, containing  $L$  items, and a fraction,  $\mu$ , select an item not currently on the list. Empirically, the turnover of items on the list has been found to be proportional to:  $L\sqrt{\mu}$  (a theoretical analysis<sup>455</sup> finds that population size does have some impact:  $L\sqrt{\mu \log(N/L)}$ ).

Studies have found that subjects are able to socially learn agreed conventions, when communicating within networks, having various topologies (containing up to 48 members; the maximum experimental condition).<sup>260</sup> One advantage of agreed conventions is a reduction in

---

<sup>vii</sup> Your author leaves a plastic beaker for the milkman to place over the bottle, otherwise, within a week the milk bottle top will be regularly opened; something that milkman new to the job have to learn.

communications effort, when performing a joint task (e.g., reduction in the number of words used<sup>301</sup>).

Transmission of information about new products Section 3.6.3...

### 3.4.3 Group learning and forgetting

The performance of groups, like that of individuals (see Figure 2.31), improves with practice; organizations also forget (in the sense that performance on a previously learned task degrades with time).<sup>764</sup> Industrial studies have focused on learning by doing,<sup>1553</sup> that is the passive learning that occurs when the same, or very similar, product is produced over time.

The impact of organizational learning during the production of multiple, identical units (e.g., a missile or airplane) can be modeled to provide a means of estimating the likely cost and timescale of producing more of the same units.<sup>586</sup> Various models of organizational production<sup>732, 1052</sup> based on connected components, where people are able to find connections between nodes that they can change to improve production, produce the power laws of organizational learning found in practice.

There are trade-offs to be made in investment in team learning; there can be improvements in performance and term performance can be compromised.<sup>230</sup>

A study by Nembhard and Osothsilp<sup>1144</sup> investigated the impact of learning and forgetting on the production time of car radios; various combined learning-forgetting models have been proposed, and the quality of fit to the data was compared. Figure 3.15 shows the time taken to build a car radio against cumulative production, with an exponential curve fitted to each period of production (break duration above the x-axis). Note, build time increases after a break and both a power law and exponential have been proposed as models of the forgetting process.

Software is easily duplicated, continual reimplementation only occurs in experiments (see Figure 2.34). Software systems also tend to be more complicated than radios and take longer to implement.

Figure 3.16 shows the man-hours needed to build three kinds of ships, 188 in total, at the Delta Shipbuilding yard, between January 1942 and September 1945. As experience is gained building Liberty ships the man-hours required, per ship, decreases.<sup>1553</sup> Between May 1943 and February the yard had a contract to build Tankers, followed by a new contract to build Liberty ships and then Colliers. When work resumes on Liberty ships, the man-hours per ship is higher than at the end of the first contract, presumably some organizational knowledge about how efficiently build this kind of ship had been lost.

When an organization loses staff having applicable experience, there is some loss of performance.<sup>52</sup> Minimum viable knowledge unit... It has been proposed<sup>681</sup> that the indigenous people's of Tasmania lost valuable skills and technologies because their effective population size shrunk when the sea level rose, cutting the islander's off from contact with mainland Australia.

An organization's knowledge and specialist skills are passed on to new employees by existing employees and established practices.

A study by Muthukrishna, Shulman, Vasilescu and Henrich<sup>1123</sup> investigated the transmission of knowledge and skills, between successive generations of teams performing a novel (to team members) task. In one experiment, each of ten generations was a new team of five people (i.e., 50 different people), with every team having to recreate a target image using GIMP. After completing the task, a generation's team members were given 15-minutes to write-up two pages of information, whose purpose was to assist the team members of the next generation. Successive teams in one sequence of ten generations, were given the write-up from one team member of the previous generation, while successive teams in another sequence of ten generations, were given the write-up from the five members of the previous generation (i.e., the experiment involved 100 people).

Figure 3.17 shows the rating given to the image produced by each team member, from each of successive generation; lines show fitted regression models. The results suggest that more information was present in five write-ups (blue-green) than one write-up (red), enabling successive generations to improve (or not).

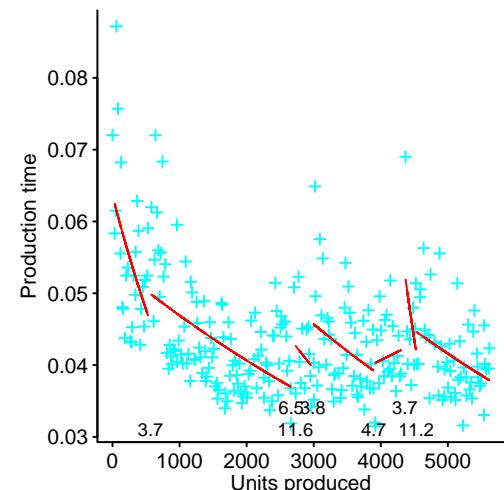


Figure 3.15: Hours required to build a car radio after the production of a given number of radios, with break periods (shown in days above x-axis); lines are models fitted to each production period. Data extracted from Nembhard et al.<sup>1144</sup> code

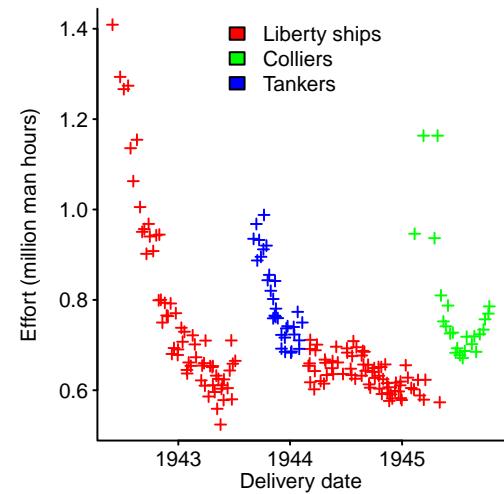


Figure 3.16: Man-hours required to build a particular kind of ship, at the Delta Shipbuilding yard, delivered on a given date (x-axis). Data from Thompson.<sup>1553</sup> code

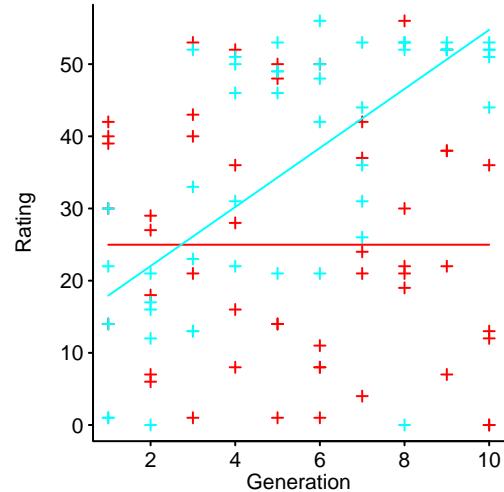


Figure 3.17: Task rating given to members of successive generations of teams; lines are a regression model fitted to the one (red) and five (blue-green) write-up generation sequences. Data from Muthukrishna et al.<sup>1123</sup> code

Outsourcing of development work, offshore or otherwise, removes the learning-by-doing opportunities afforded in-house development; in some cases management may learn from the outsource vendor.<sup>263</sup>

### 3.4.4 Information asymmetry

The term *information asymmetry* describes the situation where one party in a negotiation has access to important, applicable, information that is not available to the other parties. The information is important in the sense that it can be used to make more accurate decisions. In markets where no information about product quality is available, low quality drives out higher quality.<sup>19</sup>

Building a substantial software system involves a substantial amount of project specific information; a large investment of time is needed to acquire and understand this information. The kinds of specific information include: application domain, software development skills and know-how, and existing user work practices.

Vendors bidding to win the contract to implement a new system can claim knowledge on similar systems, but cannot claim any knowledge of a system that does not yet exist. In any subsequent requests to bid to enhance the now-existing system, one vendor can claim intimate familiarity with the workings of the software they implemented. This information asymmetry may deter other vendors from bidding, and places the customer at a disadvantage in any negotiation with the established vendor.

Figure 3.18 shows the ratio of actual to estimated effort for 25 releases of one application, over six years (release 1.1 to 9.1, figures for release 1 are not available).<sup>734</sup> Possible reasons for the estimated effort being so much higher than the actual effort include: cautiousness on the part of the supplier, and willingness to postpone implementation of features planned for the current release to a future release (one of the benefits of trust built up between supplier, and customer in an ongoing relationship is flexibility of scheduling).

The term *vaporware* is used to describe the practice of announcing software products with a shipment date well in the future. These announcements are a signalling mechanism that is underpinned by asymmetrical information, because there may or may not be a real product under development; uses include: keeping existing customer happy that an existing product has a future and deterring potential competitors.<sup>390</sup> A study by Bayus, Jain and Rao<sup>123</sup> investigated the delay between promised availability, and actual availability, of 123 software products. Figure 3.19 shows that the interval, in months, between the announcement date and promised product availability date has little correlation with the interval between promised and actual delivery date. The fitted regression line shows that announcements promising product delivery in a month or two are slightly more accurate than those promising delivery further in the future.

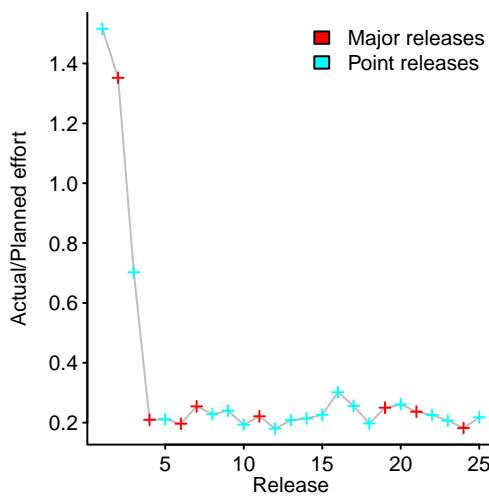


Figure 3.18: Ratio of actual to estimated hours of effort to enhance an existing product, for 25 versions of one application. Data from Huijgens et al.<sup>734</sup> [code](#)

### 3.4.5 Moral hazard

A *moral hazard* occurs when information asymmetry exists, and the more informed party has some control over the unobserved attributes; the traditional example is a manager running a business for those who own the company (the manager has opportunities to enrich himself, at the expense of the owners, who lack the information needed to detect what has happened), but it can also apply to software developers making implementation decisions, e.g., basing their choice on the enjoyment they expect to experience, rather than the likely best technical solutions.

Agency theory deals with the conflict of interests of those paying for work to be done, and those being paid to do it.

With so many benefits on offer to the cognitariate, employers need a mechanism for detecting free-riders; employees have to signal hedonic involvement, e.g., by working long hours.<sup>910</sup>

### 3.4.6 Group survival

Groups face a variety of threats to their survival, including: more productive members wanting to move on (brain drain), the desire of less productive people to join (adverse selection), and the tendency of members to shirk (e.g., social loafing).

Groups are function through the mutual cooperation, between members. Reciprocity, actions toward another involving an expectation of a corresponding cooperative response from the other party, is a fundamental component of group cohesion.

Simulations<sup>618</sup> have found that the presence of reciprocity (x helps/harms y, who in term helps/harms x), and transitivity (if x and y help each other, and y helps z, then x is likely to help z) in a uniform population, are sufficient for distinct subgroups to form; see [eco-nomics/2014+Gray+Rand.R](#).

Cooperation via *direct reciprocity* is only stable when the probability of interacting with a previously encountered member  $P_i$ , meets the condition:<sup>1184</sup>  $\frac{c}{b} < P_i$ , where:  $c$  is the cost of cooperation, and  $b$  is the benefit received (by one party if they defect, and by both parties if they both cooperate; if an altruist member does not start by cooperating, both parties receive zero benefit); cooperation via *indirect reciprocity*, where decisions are based on reputation (rather than direct experience), is only stable when the probability of knowing a members' reputation,  $P_k$ , meets the condition:<sup>1169</sup>  $\frac{c}{b} < P_k$ .

Members may be tempted to take advantage of the benefits of group membership, without making the appropriate contributions to the group, or may fail to follow group norms. If a group is to survive, its members need to be able to handle so called *free-riders*.

In an encounter between two people, either may choose to cooperate, or not, with the other. A problem known as the *prisoner's dilemma* has to dominate the analysis of interactions between two people. Encounters involving the same person can be a recurring event (known as the *iterated prisoner's dilemma*); in a noise free environment, Tit-for-Tat (Tft, start by cooperating, and then reciprocate the partner's behavior) is the best currently known solution. In practice Tft is easily disrupted by noise, e.g., responding based on imperfect recall of past interactions;<sup>1617</sup> alternative strategies include generous Tft (follow Tft, except cooperate with some probability when partner defects)... .

While the analysis of prisoner's dilemma style problems can produce insights, it may not be possible to obtain sufficiently accurate values for the variables involved. The following analysis<sup>71</sup> illustrates some of the issues.

In a commercial environment, developers may be vying with each other for promotion, or a pay rise. Developers who deliver projects on schedule are likely to be seen by management as worthy of promotion or a pay rise.

Consider the case of two software developers who are regularly assigned projects to complete, by a management specified date; with probability  $p$ , the project schedules are unachievable. If the specified schedule is unachievable, performing Low quality work will enable the project to be delivered on schedule, alternatively the developer can tell management that slipping the schedule will enable High quality work to be performed.

Performing Low quality work is perceived as likely to incur a penalty of  $Q_1$  (because of its possible downstream impact on project completion), if one developer chooses Low, and  $Q_2$ , if both developers choose Low. It is assumed that:  $Q_1 < Q_2 < C$ . A High quality decision incurs a penalty that developers perceive to be  $C$  (telling management that they cannot meet the specified schedule makes a developer feel less likely to obtain a promotion/pay-rise);

Let's assume that both developers are given a project, and the corresponding schedule; if either developer faces an unachievable deadline, they have to immediately decide whether to produce High or Low quality work. When making the decision, information about the other developer's decision is not available.

An analysis<sup>71</sup> shows that, both developers choosing High quality is a pure strategy (i.e., always the best), when:  $1 - \frac{Q_1}{C} \leq p$ , and High-High is Pareto superior to both developers choosing Low quality when:  $1 - \frac{Q_2}{C-Q_1+Q_2} < p < 1 - \frac{Q_1}{C}$ .

Obtaining reasonably estimates for  $p$ ,  $C$ , and  $Q_1$  is likely to be problematic. Inferences drawn from the forms of the equations, intended to encourage a High-High decision (e.g., be generous when schedule implementation time, and don't penalise developers when they ask for more time), could have been inferred without using a Prisoner's dilemma model.

There is experimental<sup>1197</sup> and real-life<sup>1182</sup> evidence that group members are willing to punish, at a cost to themselves, a member who fails to follow agreed group norms... .

Align incentives so that individuals benefit from acting in the group's interest. Giving social status to actions that benefit the group... .

?

### 3.4.7 Group problem solving

Group problem solving performance<sup>926</sup> can be strongly affected by the characteristics of the problem; problem characteristics include:

- unitary/divisible: is there one activity that all members have to perform at the same time (e.g., rope pulling), or can the problem be divided into distinct subcomponents and assigned to individual members to perform?
- maximizing/optimizing: is quantity of output the goal, or quality of output (e.g., a correct solution)?
- interdependency: methods of combining member outputs to produce a group output, for instance: every member complete a task, individual member outputs added together, averaged, one member's output is chosen as the group output.

When solving Eureka-type problems (i.e., a particular insight is required), the probability that a group containing  $k$  individuals will solve the problem is:  $P_g = 1 - (1 - P_i)^k$ , where:  $P_i$  is the probability of one individual solving the problem. When the solution involves  $s$  subproblems, the probability of group success is:  $P_g = \left[1 - \left(1 - P_i^{1/s}\right)^k\right]^s$ . Some studies<sup>978</sup> have found group performance having the characteristics of a combination-of-individuals model.

Studies<sup>1515</sup> have consistently found that for brainstorming problems (i.e., producing creative ideas), individuals consistently outperform groups (measured by quantity and quality of ideas generated, per person). Despite overwhelming experimental evidence, the belief that groups outperform individuals has existed for decades.<sup>viii</sup>

A study by Bowden and Jung-Beeman<sup>189</sup> investigated the time taken, by individuals, to solve word association problems. Subjects saw three words, and were asked to generate a fourth word that could be combined with each of word to produce a compound word or phrase; for instance, the words SAME/TENNIS/HEAD can all be combined with MATCH.<sup>ix</sup> The 289 subjects were divided into four groups, each with a specified time limit on generating a problem solution (2, 7, 15 and 30 seconds).

Figure 3.20 shows the percentage of individuals, in each group, who successfully generated a solution, against the mean time taken. Each plus corresponds to one of the 144 problems, with colors denoting the time limit (solution time was not collected for the group having a 2-second time limit). Lines show a sample of the corresponding performance pairs for the same problem, in each of the time-limit groups.

The results show that these problems have Eureka-type characteristics, with some problems being solved quickly by nearly all subjects, and other problems were solved more slowly by a smaller percentage of subjects.

The presence of others (e.g., spectators, or bystanders) has been found to have a small effect on an individual's performance (i.e., plus/minus 0.5% to 3%).<sup>180</sup>

The information-sampling model<sup>1495</sup> suggests that the likelihood of a group focusing on particular information increases with the number of members who are already aware of it, i.e., information known by one, or a few members, is less likely to be discussed because there are fewer people able to initiate the discussion. The term *hidden profile* is used to denote the situation where necessary task information is not widely known, and distributed over many group members.

Studies have found that as group size increases, the effort exerted by individuals decreases.<sup>824</sup> the terms *social loafing* and *Ringelmann effect* (after the researcher who first studied<sup>1335</sup> group effort, when pulling a rope), are used to describe this behavior. Social loafing has been found to occur in tasks that do not involve any interaction between group members, e.g., when asked to generate sound by clapping and cheering a group, the sound pressure generated by individuals decreases as group size increases.<sup>924</sup> An extreme form of social loafing is *bystander apathy*, such as when a person is attacked and onlookers do nothing because they assume somebody else will do something.<sup>923</sup>

<sup>viii</sup> This belief in the benefits of brainstorming groups has been traced back to a book published by an advertising executive in the 1950s.

<sup>ix</sup> Synonymy (same = match), compound word (matchhead), and semantic association (tennis match).

A study by Akdemir and Ahmad Kirmani<sup>18</sup> analyzed student performance when working in teams and individually, based on marks given for undergraduate projects. Students completed two projects working in a team and one project working alone, with all team members given the same mark for a project. Figure 3.21 shows a density plot of the difference between mean team mark and individual mark, for the 386 subjects, broken down by team size.

?

?

?

?

??

### 3.4.8 Cooperative competition

Within ecosystems driven by strong network effects, there can be substantial benefits in cooperating with competitors; the cost of not being on the winning side of the war, to set a new product standard, can be very high.<sup>1414</sup>

Software systems need to be able to communicate with each other, agreed communication protocols are required. The communication protocols may be decided by the dominant player (e.g., Microsoft with its Server protocol specifications<sup>1076</sup>), by the evolution of basic communication between a few systems to something more complicated involving many systems, or by interested parties meeting together to produce an agreed specification. The components of hardware devices also need to interoperate, and several hundred interoperability standards are estimated to be involved in a laptop.<sup>158</sup>

Various standards' bodies, organizations, consortia, and groups have been formed to document agreed-upon specifications. Committee members are often required to notify other members of any patents they have, that impinge on the specification being agreed to; organizations that failed to reveal patents they hold, when they subsequently attempt to extract royalties from companies implementing the agreed specification, may receive large fines and licensing their patents under reasonable terms may be government enforced.<sup>479</sup>

A study by Simcoe<sup>1440</sup> investigated the production of communication specifications (i.e., RFCs) by the Internet Engineering Task Force (IETF) between 1993 and 2003 (the formative years of the Internet). Figure 3.22 shows that the time taken to produce an RFC having the status of a Standard, increased as the percentage of commercial membership of the respective committee increased, but there was no such increase for RFCs not having the status of a standard; Simcoe proposed that the delay was caused by conflicts between vendors jockeying for commercial advantage...

For some applications there is a commercial incentive for companies to cooperate to build a common system, e.g., sharing in a winner take-all market, or because the benefits of owning an in-house system are not worth the costs. The projects to build these applications have teams containing developers from multiple companies....

A study by Teixeira, Robles and González-Barahona<sup>1541</sup> investigated the evolution of developers, employed by companies listed in the legend, working on the releases of the OpenStack project between 2010 and 2014. Figure 3.23 shows the involvement of top ten companies, out of over 200 organizations involved, as a percentage of employed developers working on OpenStack.

## 3.5 Company economics

The value of a company<sup>205</sup> is derived from two sources: tangible goods such as buildings, equipment and working capital, and intangible assets, which are a product of knowledge (e.g., employee know-how, intellectual property and customer switching costs); see Figure 1.8.

Governments have been relatively slow to include intangible investments in the calculation of GDP (starting in 1999 in the US and 2001 in the UK), and different approaches to valuing

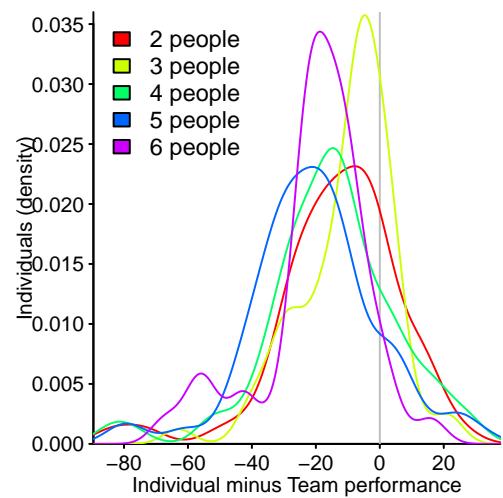


Figure 3.21: Density plot of the difference between mean team mark and individual mark, broken down by team size. Data from Akdemir et al.<sup>18</sup> [code](#)

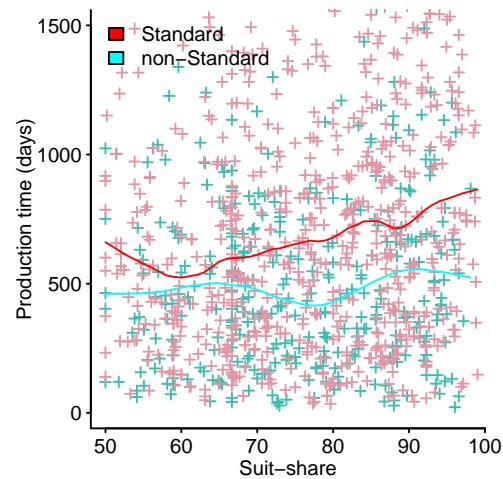


Figure 3.22: Loess fits to time taken to publish an RFC having Standard or non-Standard status, for IETF committees having a given percentage of commercial membership (people wearing suits). Data from Simcoe.<sup>1440</sup> [code](#)

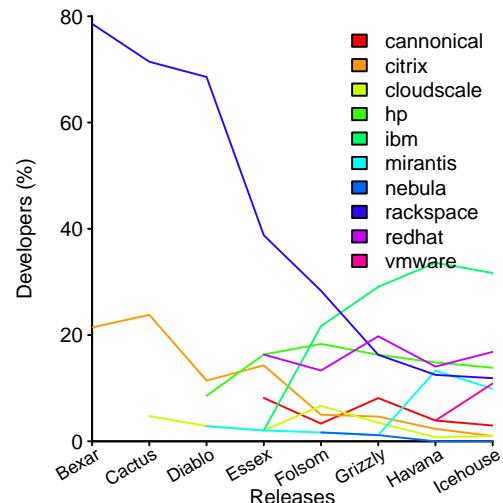


Figure 3.23: Percentage of developers, employed by given companies, working on OpenStack at the time of a release (x-axis). Data from Teixeira et al.<sup>1541</sup> [code](#)

software<sup>16</sup> has led to an increase in uncertainty when comparing country GDP (McGee<sup>1045</sup> discusses accounting practices, for software, in the UK and US during the 1970s early 1980s). The accounting treatment of intangibles depends on whether it is purchased from outside the company, requiring it to be treated as an asset, or generated internally where it is often treated as an expense.<sup>1114</sup> A company's accounts may only make sense when its intangible assets are included in the analysis.<sup>737</sup>

The financial structure of even relatively small multinational companies, is likely to be complicated.<sup>883</sup>

### 3.5.1 Cost accounting

The purpose of cost accounting is to help management make effective cost control decisions. In traditional industries the primary inputs are the cost of raw materials (e.g., the money needed to build the materials needed to build a widget), and labor costs (e.g., the money paid to the people involved in converting the raw materials into a widget); other costs might include renting space where people can work and the cost of consumables such as electricity.

The production of software systems is people driven, and people are the primary cost source.

The total cost of a software developer can be calculated from the money they are paid, plus any taxes levied by the government, on an employer, for employing somebody (e.g., national insurance contributions in the UK<sup>x</sup>); the term *fully loaded cost* is sometimes used.

Systems such as activity-based costing provide detailed information about the cost of providing every product and service, provided every employee keeps a complete record of the time spent on their activities...

Without public data it is not possible to analyse the extent to which software development contains activities that are repeated on a regular basis, and can be tuned using cost accounting techniques...

Revenue per employee... Meti data... Does not include contractors, consultants...

### 3.5.2 The shape of money

Money is divided up, and categorized, in various ways for a variety of different reasons. Figure 3.24 illustrates the way in which UK and US tax authorities require registered companies to apportion their income to various cost centers.

Within a company, the most senior executives allocate a budget for each of the organizational groups within the company. These groups, in turn, divide up their available budget between their own organizational groups, and so on. This discrete allocation of funds can have an impact on how software development costs are calculated.

Take the example of a development project, where testing is broken down into two phases, i.e., integrating testing and acceptance testing, each having its own budget, say  $B_i$  and  $B_a$ .

One technique sometimes used for measuring the cost of faults is to divide the cost of finding them (i.e., the allocated budget), by the number of faults found. For instance, if 100 faults are found during integration testing, the cost per fault in this phase is  $\frac{B_i}{100}$ , and if five faults are found during acceptance testing, the cost per fault in this phase is  $\frac{B_a}{5}$ .

One way of reducing the cost per fault found during acceptance testing would be to reduce the effectiveness of integration testing. Because, for a fixed budget, the cost per fault decreases as the number of faults found increases.

This accountancy-based approach to measuring the cost of faults, creates the impression that it is more costly to find faults later in the process, compared finding them earlier.<sup>185</sup> A conclusion created by the artefact of a fixed budget, and the typical situations where fewer faults are found later in the development process.

Time taken to fix faults may less susceptible to accounting artefacts, provided the actual time is used (i.e., not the total allocated time). Figure 3.25 shows the average number of days used to fix a defect detected in a given phase (x-axis), that had been introduced in an earlier phase (colored lines), based on 38,120 defects in projects at Hughes Aircraft.<sup>1671</sup>

<sup>x</sup> At the time of writing this was zero-rated up to a threshold, then 12% of employee earnings, increasing on reaching an upper threshold.

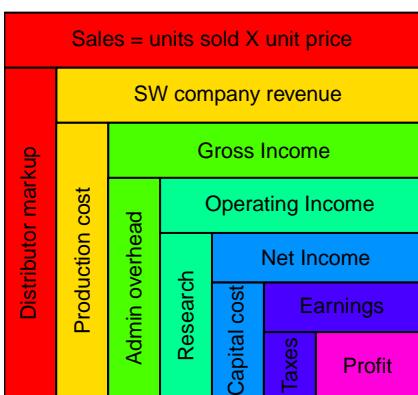


Figure 3.24: Accounting practice for breaking down income from sales... [code](#)

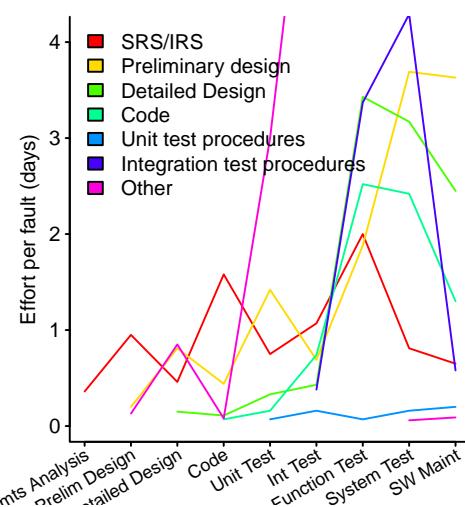


Figure 3.25: Average effort (in days) used to fix a defect detected in a given phase (x-axis) that had been introduced in an earlier phase (colored lines) in projects at Hughes Aircraft. Data extracted from Willis et al.<sup>1671</sup> [code](#)

### 3.5.3 Valuing software

Like any other item, if no one is willing to pay money for the software, it has zero sales value. The opportunity cost of writing software from scratch, along with the uncertainty of being able to complete the task, and the undocumented business rules it implements, is what makes it possible for existing software to be worth significantly more than the original cost of creating it. For accounting purposes, software may be valued in terms of the cost of producing it.

Various approaches to valuing software, and any associated intellectual property are available.<sup>1661</sup> A simplistic approach to estimating production cost is to use the quantity of code (often measured in lines) as a proxy. This approach ignores the opportunity and risk costs associated with the original production, along with the domain specific expertise that was involved.

A study by Gayek, Long, Bell, Hsu and Larson<sup>551</sup> obtained general effort/size information on 452 military/space projects. Figure 3.26 shows the lines of code, and the developer effort (in months) used to create them. There is an order of magnitude variation in developer effort (i.e., costs) for the same ESLOC, and the ESLOC vs. Effort relationship varies between end-user application domains.

Commercial companies are required to keep accurate financial accounts, whose purpose is to provide essential information for those with a financial interest in the company, including governments seeking to tax profits. Official accounting organizations have created extensive, and ever-changing, rules for producing company accounts, including methods for valuing software and other products of intellectual effort.<sup>676</sup> In the US, the Financial Accounting Standards Board has issued an accounting standard “FASB Codification 985-20” (FAS 80<sup>496</sup> was used until 2009) covering the “Accounting for the Costs of Computer Software to Be Sold, Leased, or Otherwise Marketed”. This allows software development costs to be treated either as an expense or capitalized (i.e., treated like the purchase of an item of hardware). An expense is tax-deductible in the financial year in which it occurs, but the software does not appear in the company accounts as having any value; the value of a capitalized item is written down over time (i.e., a percentage of the value is tax-deductible over a period of years), but has a value in the company accounts.<sup>844</sup>

The decision on how software development costs appear in the company accounts can be driven by the desire to project a certain image to interested outsiders (e.g., the company is worth a lot because it owns valuable assets<sup>4</sup>), or to minimise tax liabilities. A study by Mulford and Roberts<sup>1115</sup> of 207 companies (primarily industry classification SIC 7372) in 2006, found that 30% capitalized some portion of their software development, while a study by Mulford and Misra<sup>1114</sup> of 100 companies in 2015, found 18% capitalizing their development.

Software made available under an Open Source license may be available at no cost, but a value can be expressed in terms of replacement cost, or the cost of alternatives...

Value based on expected lifetime... lifetime of SQL tables discussed in Section 7.5.1..

## 3.6 Maximizing ROI

Paying people to write software is an investment, and investors want to maximise the return on their investments.

Some people write software to acquire non-monetary income, and unless this income is derived purely from the creation process, maximising returns will be of interest.

To be viable, a commercial product require a market capable of paying what it takes. It is unwise to invest in software production, without being confident that the target market is capable of producing the desired return on investment.

One study<sup>838</sup> of Android Apps found that 80% of reviews were made by people owning a subset of the available devices (approximately 33%). Given the cost of testing an App in the diverse Android ecosystem, the ROI may be maximized by ignoring those devices that are owned by a small percentage of the customer base.

Accounting issues around choices such as purchase vs. leasing, contractors vs. employees, and making efficient use of income held offshore,<sup>159</sup> are outside the scope of this book.

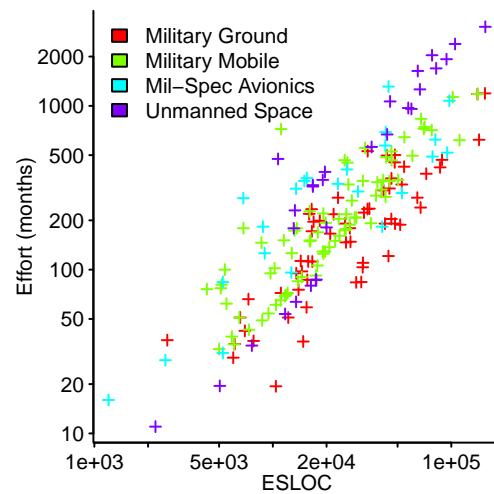


Figure 3.26: Months of developer effort needed to produce systems containing a given number of lines of code, for various application domains. Data from Gayek et al.<sup>551</sup> code

### 3.6.1 Value creation

Many businesses structure their value creation processes around the concept of a value chain; in some software businesses value creation is structured around the concept of a value network.<sup>1485</sup>

A *value chain* is the collection of activities that transform inputs into products. “A firm’s value chain and the way it performs individual activities are a reflection of its history, its strategy, its approach to implementing its strategy, and the underlying economics of the activities themselves.”<sup>1265</sup>

A value chain used for bespoke software development might be derived from the development methodology used to produce a software system, e.g., the phases of the waterfall model.<sup>177</sup>

The business model for companies selling software products would include a value chain that included marketing, customer support and product maintenance.

Software business models<sup>1383</sup> ...

and the data... .

In a two-sided market, value flows from one side to the other... Companies create value by facilitating a network relationship between their customers... .

A company whose business is solving customer problems (e.g., professional services) might be called a *value shop*.<sup>1485</sup>

### 3.6.2 Product/service pricing

Vendors can set whatever price they like, for their product or service. The ideal is to set a price that maximises profit; there is no guarantee that income from sales will cover the investment made in development and marketing. Even given extensive knowledge of potential customers and competitors, setting prices is very difficult.<sup>1132, 1436</sup>

Like everything else, software products are worth what customers are willing to pay. When prices quoted by vendors are calculated on an individual customer basis, inputs considered include how much the customer might be able pay.<sup>1386</sup> Shareware, software that is made freely available in the hope that users will decide it is worth paying for,<sup>1531</sup> is one of the purest forms of customer payment decision-making.

Customer expectation, based on existing related products, acts as a useful ballpark to start the estimation process; Figure 3.27 illustrates the relative performance pricing used by Intel. It may be possible to charge more for a product that provides more benefits, or perhaps the price has to match that of the established market leader, and these benefits are used to convince existing customers to switch.

Figure 3.30 shows the prices charged by several established C/C++ compiler vendors. In 1986<sup>xi</sup> Zorland entered the market with a £29.95 C compiler, an order of magnitude less than what most other vendors were charging at the time. This price was low enough for many developers to purchase a copy out of company petty cash, and Zorland C quickly gained a large customer base. Once the product established a quality reputation, Zorland were able to increase prices to the same level as those charged by other major vendors (Zorland sounds very similar to the name of another major software vendor of the period, Borland; letters from company lawyers are hard evidence that somebody in authority thinks your impact on their market is worth investing money on. Zorland became Zortech).<sup>xii</sup>

Many companies give managers the authority to purchase low cost items, with the numeric value of low price increasing with seniority. The upper bound for the maximum price that can be charged for a product or service, that can be sold relatively quickly to businesses, is the purchasing authority of the target decision maker. Once the cost of an item reaches some internally specified value (perhaps a few thousand pounds or dollars), companies require a more bureaucratic purchase process to be followed, perhaps involving a purchasing committee, or even the company board. Navigating a company’s bureaucratic processes requires a

<sup>xi</sup> Richard Stallmann’s email announcing the availability of the first public release of gcc was sent on 22 March 1987.

<sup>xii</sup> Being in the compiler business your author had copies of all the major compilers, and Zorland C was the compiler of choice for a year or two. Other low price C compiler vendors were unable to increase their prices because of quality issues relative to other products on the market, e.g., Mix C.

sales rep, and a relatively large investment of time, increasing the cost of sales, and requiring a significant increase in selling price; a price dead-zone exists between the maximum amount managers can independently sign-off on, and the minimum amount it is worth selling via sales reps.

Supply and demand is a commonly cited economic pricing model. The supply curve is the quantity of an item that a supplier is willing, and able, to supply (i.e., sell) to customers at a given price, and the demand curve is the quantity of a product that will be in demand (i.e., bought by customers) at a given price. If these two curves intersect, the intersection point gives the price/quantity at which suppliers and customers are willing to do business; see Figure 3.28.

Events can occur that cause either curve to shift. For instance, the cost of manufacturing an item may increase/decrease, shifting the supply curve up/down on the price axis; or customers may find a cheaper alternative, shifting the demand curve down on the price axis.

In some established markets, sufficient historic information has accumulated for reasonably accurate supply/demand curves to be drawn. Predicting the impact of changing circumstances on supply-demand curves, remains a black art for many products and services.

Software is a relatively new market and one that continues to change relatively quickly. This changeability makes estimating supply/demand curves for software products little more than wishful thinking.

Product pricing has an effect on the volume of products sold...

Other pricing issues include site licensing, discounting and selling through third-parties.<sup>1419</sup> However, the lack of data prevents these issues being discussed here.

Listed prices are often slightly below a round value, e.g., £3.99 or £3.95 rather than £4.00. People have been found to perceive this kind of price difference to be greater than the actual numerical difference<sup>1550</sup> (the value of the left digit, and the numeric distance effect have been used to explain this behavior). Figure 3.29 shows the impact visually distinct, small price differences, have on the rate of sale of items listed on Gumroad (a direct to consumer sales website). Other consumer price perception effects include precise prices vs. round prices<sup>1551</sup> ...

The price of a basket of common products, over time, is used to calculate the consumer price index, and changes in product prices over time can be used to check the accuracy of official figures.<sup>1213</sup> Products may evolve over time, with new features being added, and existing features updated; techniques for calculating quality adjusted prices have been developed,<sup>1519</sup> see Figure 3.27.

A study by Viard<sup>1608</sup> investigated software retail and upgrade pricing, in particular C and C++ compilers available under Microsoft DOS and Windows, between 1987 and 1998. Figure 3.30 shows that the retail price of C/C++ compilers, running under Microsoft DOS and Windows, had two bands that remained relatively constant. Microsoft had a policy of encouraging developers to create software for Windows,<sup>7</sup> on the basis that significantly greater profits would be made through volume sales of Windows, compared to sales of compilers. Microsoft's developer friendly policy kept the price of C/C++ compilers under Windows down, compared to other platforms.<sup>xiii</sup>

### 3.6.3 Predicting sales volume

The likely volume of sales is a critical question for any system intended to be sold to multiple customers.

When one product substitutes another, new for old, market share of the new product is well fitted by a logistic equation,<sup>502</sup> whose maximum is the size of the existing market. Software may be dependent on the functionality provided by the underlying hardware, which may be rapidly evolving.<sup>859</sup> Figure 3.31 shows how software sales volume lags behind sales of the hardware needed to run it. An installed hardware base can be an attractive market for software.<sup>1016</sup>

Estimating the likely sales volume for a product intended to fill a previously unmet customer need, or one that is not a pure substitution, is extremely difficult (if not impossible).

<sup>xiii</sup> The 1990s was the decade in which gcc, the only major open source C/C++ compiler at the time, started to be widely used.

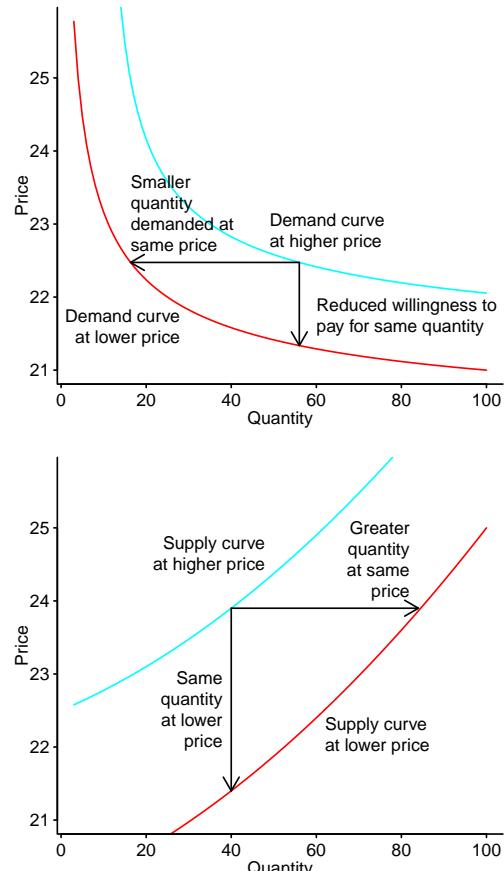


Figure 3.28: Example supply and demand curves. [code](#)

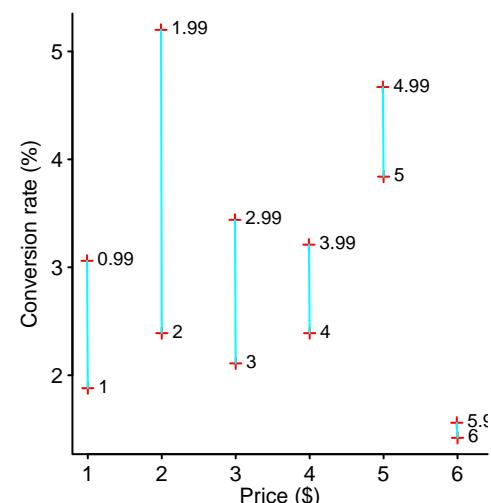


Figure 3.29: Rates at which product sales are made on Gumroad at various prices; lines join prices that differ in 1¢, e.g., \$1.99 and \$2. Data from Nichols.<sup>1155</sup> [code](#)

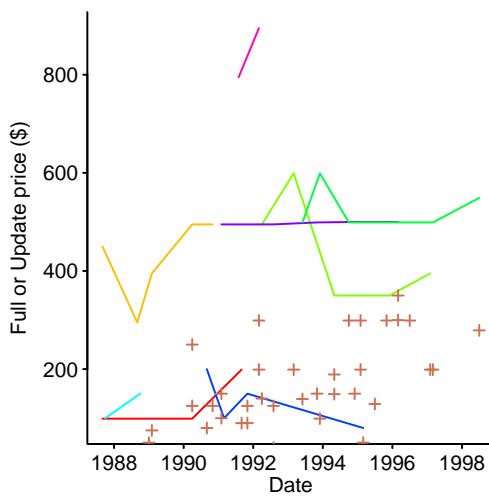


Figure 3.30: Vendor C and C++ compiler retail price (different line for each product), and upgrade prices (pluses) for products available under MS-DOS and Microsoft Windows between 1987 and 1998. Data kindly provided by Viardi.<sup>1608</sup> [code](#)

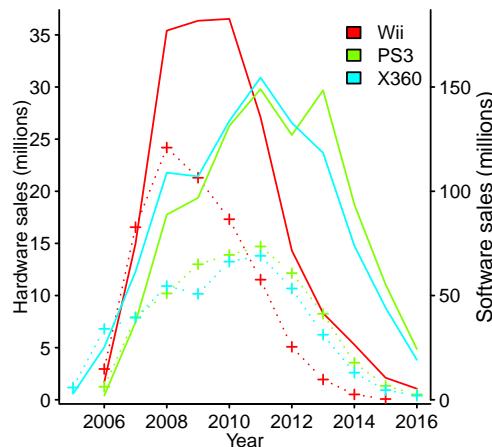


Figure 3.31: Sales of game software (solid lines) for the corresponding three major seventh generation hardware consoles (dotted lines). Data from VGChartz.<sup>1607</sup> [code](#)

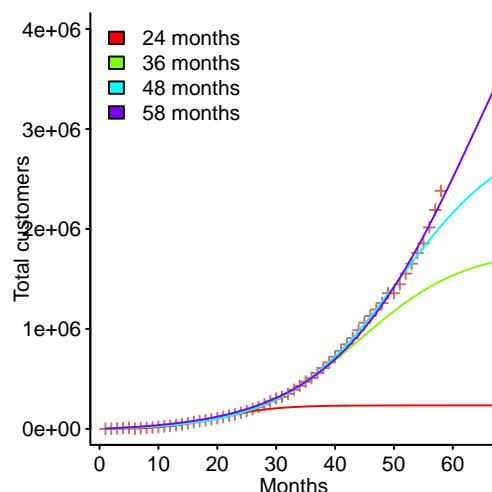


Figure 3.32: Growth of GitHub users during its first 58 months. Data from Irving.<sup>757</sup> [code](#)

The Bass model<sup>112, 1227</sup> models customer adoption of a new product, and successive releases (the following analysis deals with the case where customers are likely to make a single purchase; the model can be extended to handle repeat sales<sup>2</sup>). The model divides customers into innovators, people who are willing to try something new, and imitators, people who buy products they have seen others using (this is a diffusion model). The interaction between innovators, imitators and product adoption, at time  $t$ , is given by the following relationship:

$$\frac{f(t)}{1 - F(t)} = p + qF(t)$$

where:  $F(t)$  is the fraction of those who will eventually adopt (i.e., have purchased) by time  $t$ ,  $f(t)$  is the probability of purchase at time  $t$  (i.e., the derivative of  $F(t)$ ,  $f(t) = dF(t)/dt$ ),  $p$  the probability of a purchase by an innovator, and  $q$  the probability of a purchase by an imitator.

This non-linear differential equation<sup>xiv</sup> has the following exact solution, for the cumulative number of adopters (up to time  $t$ ), and the instantaneous number of adopters (at time  $t$ ):

$$F(t) = \frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p}e^{-(p+q)t}} \quad \text{and} \quad f(t) = \frac{(p+q)^2}{p} \frac{e^{-(p+q)t}}{\left[1 + \frac{q}{p}e^{-(p+q)t}\right]^2}$$

Actual sales, up to, or at, time  $t$ , are calculated by multiplying by  $m$ , the total number of product adopters.

When innovators dominate (i.e.,  $q \leq p$ ), sales decline from an initial high-point; when imitators dominate (i.e.,  $q > p$ ), peak sales of  $m\left(\frac{1}{2} - \frac{p}{2q}\right)$  occurs at time  $\frac{m}{p+q} \log \frac{q}{p}$ , before declining; the expected time to adoption is:  $E(T) = \frac{m}{q} \log \frac{p+q}{p}$ .

The exact solution applies to a continuous equation, but in practice sales data is discrete (e.g., monthly, quarterly, yearly). In the original formulation the model was reworked in terms of a discrete equation, and solved using linear regression (to obtain estimates for  $p$ ,  $q$ ); however, this approach produces biased results. Benchmarking various techniques<sup>1227</sup> finds that fitting the above non-linear equation to the discrete data produces the least biased results.

Vendors want reliable estimates of likely sales volume as early in the sales process as possible, but building accurate models requires data covering a non-trivial range of the explanatory variable (time in this case). Figure 3.32 shows the number of Github users during its first 58 months, and Bass models fitted to the first 24, 36, 48 and 58 months of data.<sup>xv</sup>

The Bass model includes just two out of the many possible variables that could affect sales volume, and models that include more variables have been developed.<sup>1089</sup> Intel have used an extended Bass Model to improve forecasting of design wins.<sup>1682</sup>

The Bass model can be extended to handle successive, overlapping generations of a product; the following example is for two generations:<sup>1167</sup>

$$S_1(t) = F_1(t)m_1 - F_2(t - \tau_2)F_1(t)m_1 = F_1(t)m_1(1 - F_2(t - \tau_2)) \\ S_2(t) = F_2(t - \tau_2)(m_2 + F_1(t)m_1)$$

where:  $S_i(t)$  are all sales up to time  $t$  for product generation  $i$ ,  $m_i$  the total number who adopt generation  $i$ , and  $\tau_2$  the time when the second generation can be bought;  $p_i$  and  $q_i$  are the corresponding purchase probabilities for each generation.

The Bass model uses two of the factors driving product sales, other factors that can play a significant role include advertising spend, and variability in market size caused by price changes. Monte Carlo simulation can be used to model the interaction of these various factors.<sup>1590</sup>

How much value, as perceived by the customer, does each major component add to a system? A technique for obtaining one answer to this question, is Hedonic regression (this approach is used to calculate the consumer price index); this fits a regression model to data on product price, and configuration data. A study by Stengos and Zacharias<sup>1499</sup> performed a hedonic analysis of the Personal Computer market, based on data such as price, date, CPU frequency, hard disc size, amount of RAM, screen width and presence of a CD; see [economics/0211\\_Computers.R](#) for details.

The ease with which software can be copied makes piracy an important commercial issue. Studies<sup>574, 1641</sup> have extended the Bass model to include the influence of users of pirated

<sup>xiv</sup> It has the form of a Riccati equation.

<sup>xv</sup> Section 10.5.1 provides further evidence that predictions outside the range of data used to fit a model can be very unreliable.

software, on other people's purchasing decisions (e.g., Word processors and Spreadsheet programs between 1987 and 1992). The results, based on the assumptions made by the models, and the data used, suggest that around 85% of users run pirated copies; see [economics/MPRA](#). The Business Software Alliance calculates piracy rates by comparing the volume of software sales against an estimate of the number of computers in use, a method that has a high degree of uncertainty because of the many assumptions involved<sup>1258</sup> (see [economics/piracyHICSS – 2010.R](#)).

In some ecosystems, e.g., mobile, many applications are only used for a short period, after they have been installed; see Figure 4.40.

In some markets most sales are closed just before the end of each yearly sales quarter, e.g., enterprise software. A study by Larkin<sup>920</sup> investigated the impact of non-linear incentive schemes<sup>xvi</sup> on the timing of deals closed by salespeople, whose primary income came from commission on the sales they closed. These accelerated commission schemes create an incentive for salespeople to book all sales in a single quarter.

Figure 3.33 shows the number of deals closed by week of the quarter, and the average agreed discount. Reasons for the significant peak in the number of deals closed at the end of the quarter include salespeople gaming the system to maximise commission and customers holding out for a better deal. Larkin concluded that the peak was salesperson driven.

### 3.6.4 Managing customers as investments

Acquiring new customers can be very expensive, and it is important to maximise the revenue from those that are acquired.

A person who uses a product without paying for it may be valued as a user. Facebook values its users (whose eyeballs are the product that Facebook sells to its customers: advertisers) using ARPU, defined as "... total revenue in a given geography during a given quarter, divided by the average of the number of MAUs in the geography at the beginning and end of the quarter."<sup>xvii</sup> Figure 3.34 shows ARPU, and cost of revenue per user (the difference is one definition of profit, or loss).

What is the total value of a customer to a company?

If a customer makes regular payments of  $m$ , the *customer lifetime value* (CLV) is given by (assuming the payment is made at the end of the period; simply add  $m$  if payment occurs at the start of the period):

$$CLV = \frac{mr}{(1+i)} + \frac{mr^2}{(1+i)^2} + \frac{mr^3}{(1+i)^3} + \dots = m \frac{r}{1-r+i} \left[ 1 - \left( \frac{r}{1+i} \right)^n \right]$$

where:  $r$  is the customer retention rate for the period,  $i$  the interest rate for the period, and  $n$  is the number of payment periods. As  $n \rightarrow \infty$ , this simplifies to:  $CLV = m \frac{r}{1-r+i}$ .

In the business world, annual maintenance agreements are a common form of regular payment, another is the sale of new versions of the product to existing customers (often at a discount).

Before renewing their maintenance agreement, customers have to see a worthwhile benefit. Possible benefits include promises of new product features, and support with helping to fix issues encountered by the customer. Vendor's need, to regularly be able to offer product improvements, means it is not in their interest to include too many new features, or fix too many open issues, in each release; something always has to be left for the next release. and no data...

### 3.6.5 Hedonism

Many open source projects do not pay any developers for contributing time and effort (a few projects include both paid and unpaid developers),

<sup>xvi</sup> The percentage commission earned in a non-linear scheme depends on the total value of sales booked in the current quarter, increasing at specified points, e.g., a salesperson booking \$250,000 in a quarter earns 2% commission, while a salesperson booking over \$6 million earns a commission of 25%; that first \$250,000 earns the first salesperson a commission of \$5,000, while it earns the second salesperson \$62,500.

<sup>xvii</sup> ARPU—Average Revenue Per User, MAU—Monthly Average Users.

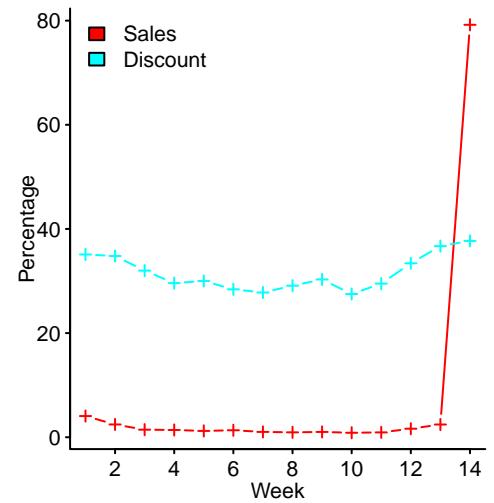


Figure 3.33: Percentage of sales closed in a given week of a quarter, with average discount given. Data from Larkin<sup>920</sup> code

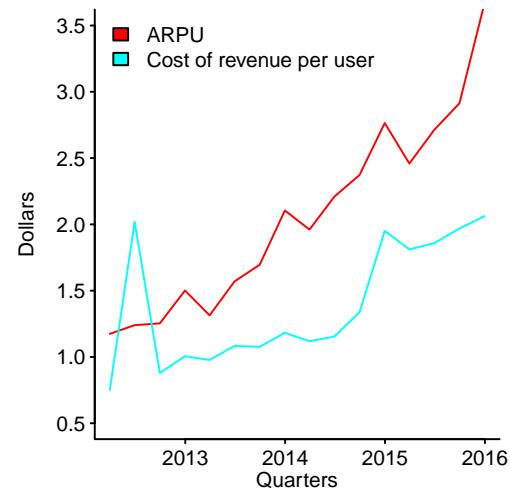


Figure 3.34: Facebook's ARPU and cost of revenue per user. Data from Facebook's 10-K filings.<sup>464,465</sup> code

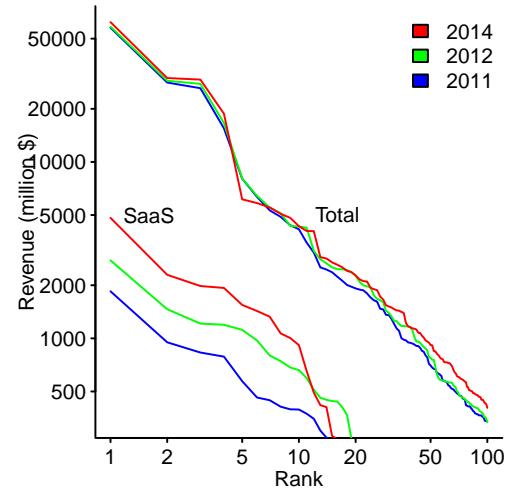


Figure 3.35: Top 100 software companies ranked by total revenue (in millions of dollars) and ranked by Software-as-a-Service revenue. Data from PwC.<sup>1291,1293</sup> code

What are the timeframes of personal involvement in an Open Source project?... TODO email...?

the Maker community<sup>7</sup> builds hardware, which often requires software to drive it...

see Section 11.3.5 for an analysis of fun and time spent working on open source projects...

### 3.6.6 Commons-based peer-production

The visibility of widely used, open source, applications, created by small groups of individuals, continues to inspire others to freely invest their energies creating or evolving software systems.

The quantity of open source software that is good enough for many commercial uses, has made the support and maintenance of some applications a viable business model.

Code distributed under an Open source license is often created using a non-contract form of production<sup>142</sup>...

Some commercial companies relicense software they have developed internally under an open source license, and may spend significant amounts of money paying employees to work on open source software. Reasons for companies appearing to waste shareholder money include:

- driving down the cost of complementary products (i.e., products used in association with the vendor's product, but are not substitutes that compete), this reduces the total cost to the customer, increasing demand, and making it more difficult for potential competitors to thrive.<sup>203</sup> Methods for driving down costs include supporting the development of free software, and ensuring there is a competitive market for the complementary product,
- giving software away to make money on the hardware that uses it,
- control of important functionality, e.g., Apple is the primary financial supporter of the LLVM compiler chain, which is licensed under the University of Illinois/NCSA open source license (which does not require contributors to supply the source code of any changes they make; unlike the GNU licensed GCC compilers)...

What are the currencies of commons-based peer-production?...

The income stream for some applications comes from advertising, that runs during program execution. A study by Shekhar, Dietz and Wallach<sup>1421</sup> investigated advertising libraries used by 10,000 applications in the Android market, and the Amazon App Store, during 2012. Figure 3.36 shows the number of apps containing a given number of advertising libraries; fitted by a Negative Binomial distribution.

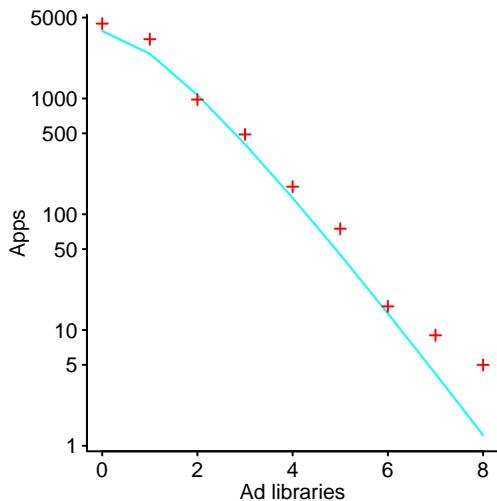


Figure 3.36: Number of applications in the Android market and Amazon App Store, during 2012, containing a given number of advertising libraries (line is a fitted Negative Binomial distribution). Data from Shekhar et al.<sup>1421</sup> code

# Chapter 4

## Ecosystems

### 4.1 Introduction

Customer demand motivates the supply of energy that drives software ecosystems.

The analysis of ecosystems can be approached from various perspectives: the population-community view of ecosystems is based on networks of interacting populations, while a process-functional view is based on the processing of resources across functional components.

Software vendors have an interest in understanding ecosystems for a variety of reasons, including wanting: to estimate the expected lifetime of their products, an estimate of the size of the potential market for their products and services, and to understand the community dynamics behind the exchange of resources.

Governments have an interest in consumer well-being, which gives them an interest in the functioning of ecosystems with regard to the cooperation and competition between vendors operating within an ecosystem.

While software ecosystems share some of the characteristics of biological ecosystems, there are some significant differences, including:

- Software can be perfectly replicated with effectively zero cost, any evolutionary pressure comes from changes in the world in which the software is operated,
- like human genes, software needs people in order to replicate, but unlike genes software is not self-replicating; people replicate software when it provides something they want, and they are willing to invest in its replication, and perhaps even its ongoing maintenance,
- software has an unlimited lifetime, in theory, in practice many systems have dependencies on the ecosystem in which they run, e.g., third-party libraries and hardware functionality.
- resource interactions are based on agreement, while in biological ecosystems creatures go to great lengths to avoid some resource interactions (e.g., being eaten).

Customer demand for software systems takes many forms, including: a means of reducing costs, creating new products sold for profit, tools that people can use to enhance their daily life, or experiencing the pleasure of creating something (e.g., writing open source). Customer demand cannot always be supplied at a profit, and the demand for new products is often insufficient for them to be profitable.<sup>i</sup>

The initial customer demand for computer systems followed the pattern seen during the industrial revolution,<sup>24</sup> with established industries seeking to reduce their labor costs invested in new technologies;<sup>1014</sup> the creation of new industries based around the use of computers came later. The first computers were built from components manufactured for other purposes

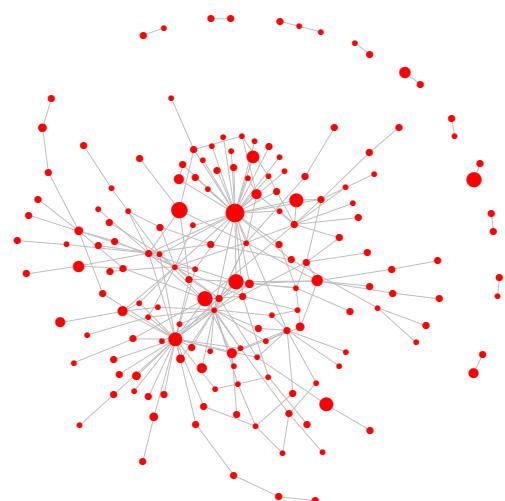


Figure 4.1: Connections between the 164 companies with Apps included in the Microsoft Office365 Marketplace (Microsoft not included); vertex size is an indicator of the number of Apps a company has in the Marketplace. Data kindly provided by van Angeren.<sup>1585</sup> [code](#)

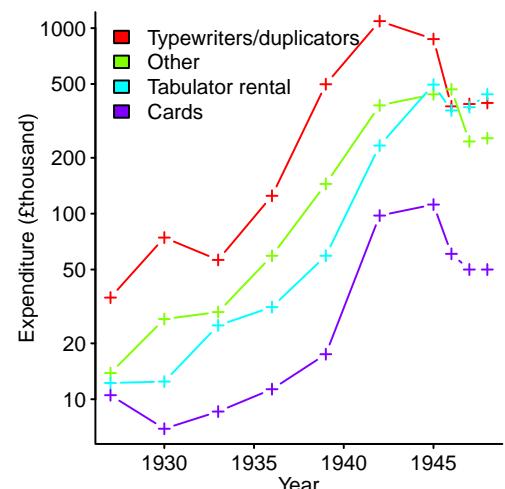


Figure 4.2: Yearly expenditure on punched cards, and tabulating equipment by the UK government. Data from Agar.<sup>9</sup> [code](#)

<sup>i</sup> The first hand-held computer was introduced around 1989, and vendors regularly reintroduced such products, claiming that customer demand now existed. Mobile phones filled a large enough demand that customers were willing to carry around an electronic device that required regular recharging; phones provided a platform for mobile computing that had a profitable technical solution.

(e.g., radio equipment), as the market for computers grew, it became profitable to manufacture bespoke devices.

The data processing and numerical calculation ecosystems<sup>627,715</sup> existed prior to the introduction of electronic computers; mechanical computers were used. Figure 4.2 shows UK government yearly expenditure on punched cards, and mechanical data processing devices.

Customer demand for solutions to the problems now solved using software may have always existed; the availability of software solutions had to wait for the necessary hardware to become available at a price that could be supplied profitably (see Figure 1.1). Significant improvements in hardware performance meant that many customers found it cost effective to regularly replace existing computer systems; Figure 4.3 shows the growth in world-wide DRAM shipments, by memory capacity shipped. Incremental improvements in technology made it possible to manufacture a greater capacity device, with many applications often constrained by memory capacity there was customer demand for greater memory capacity, and sales funded the next incremental improvements.

Who is the customer, and, which of their demands can be most profitably supplied? These tough questions are entrepreneurial and marketing problems, and not the concern of this book.

The analysis in this book is oriented towards those involved in software development, rather than their customers, however it is difficult to completely ignore the dominant supplier of the energy (e.g., money) needed for software ecosystems to continue to exist. The three primary ecosystems discussed in this chapter are oriented towards customers, vendors (i.e., companies engaged in economic activities, such as selling applications), and developers (i.e., people having careers in software development).

The firestorm of continual displacement of existing systems has kept the focus of practice on development of new systems, and major enhancements to existing systems. Post-firestorm, the focus of practice in software engineering is as an infrastructure discipline.<sup>1574</sup>

Software ecosystems have been rapidly evolving for many decades, and change is talked about as-if it were a defining characteristic, rather than a phase that ecosystems go through on the path to relative stasis. Change is so common that it has become blasé, the mantra has now become that existing ways of doing things must be disrupted. The real purpose of disruption is redirection of profits, from incumbents to those financing the development of systems intended to cause disruption. The fear of being disrupted by others is an incentive for incumbents to disrupt their own activities. Only the paranoid survive<sup>633</sup> is a mantra for those striving to get ahead in a rapidly changing ecosystem.

The first release of a commercial software project implements the clients' view of the world, from an earlier time. In a changing world, unimportant software systems have to be adapted, if they are to continue to be used; important software systems force the world to be adapted to make use of them.

A study by van Angeren, Alves and Jansen<sup>1585</sup> investigated company and developer connections in several commercial application ecosystems. Figure 4.1 shows the connections between the 164 companies in the Microsoft Office365 Apps Marketplace (to be included, Apps have to meet platform added-value requirements).

Figure 4.4 illustrates how one company, IBM, dominated the first 30+ years of the computer industry.

Major geopolitical regions have distinct customer ecosystems, providing opportunities for region specific software systems to become established (e.g., Brazilian developer response to the imposition of strong trade barriers<sup>746</sup>).

The choice of perspective used for ecosystem analysis may be driven by the available data, e.g., it may be easier to obtain measure populations than resource usage...

### 4.1.1 Funding

The continued viability of a software ecosystem is dependent on funding (which may take the form of money or volunteer effort).

Organizations may employ developers for in-house work, with funding coming from the products and services the organization provides. For large projects an organization may outsource development to companies who specialise in such work.

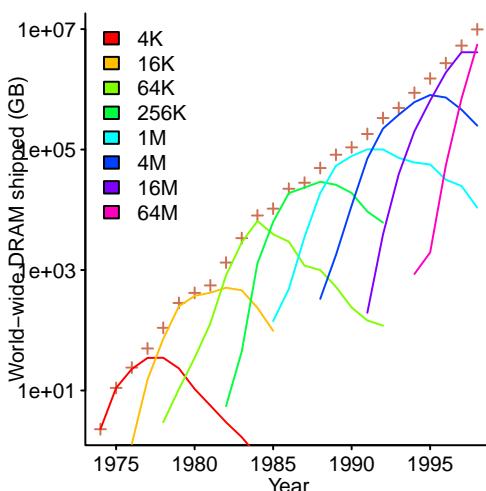


Figure 4.3: Total gigabytes of DRAM shipped world-wide in a given year, stratified by device capacity (in bits). Data from Victor et al.<sup>1610</sup> code

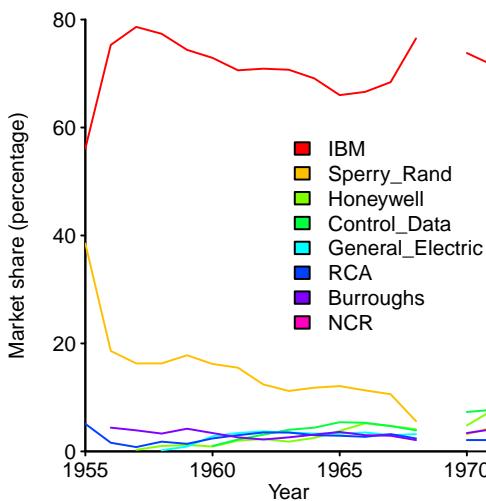


Figure 4.4: Computer installation market share of IBM, and its top seven competitors (known at the time as the seven dwarfs; no data is available for 1969). Data from Brock.<sup>213</sup> code

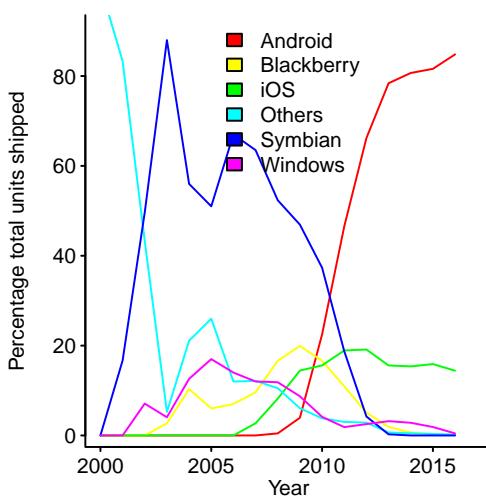


Figure 4.5: Mobile phone operating system shipments, as percentage of total per year. Data from Reimer<sup>1322</sup> (before 2007), and Gartner<sup>548</sup> (after 2006). code

Investors fund startups to create products or services that contain software components...

Software startups take many forms, from a pseudo-independent group working within a large company, to a few people starting a new company. Reasons for forming a startup vary from wanting to make a living running a business, to being funded by Venture Capital (VC) to the tune of many millions (Martínez<sup>1024</sup> discusses working in a VC funded company).

Exit strategies (extracting the profit from investments made in a company) used by VCs include: selling startups to a listed company (reasons for large companies to buy startups range from acquiring people who have specific skills,<sup>340</sup> removing potential future competitors, and the acquired company becoming a profit center; see Figure 4.6) and an IPO (i.e., having the company's shares publicly traded on a stock exchange; between 2011 and 2015 the number of software company IPOs was in the teens and for IT services and consulting companies the number was in single digits<sup>1294</sup>). Venture capitalists are typically paid a 2% management fee on committed capital, and a 20% profit-sharing structure;<sup>1154</sup> the VCs are making money, while those who invested in VCs over the last 20-years could have received greater returns by investing in a basket of stocks in the public exchanges.<sup>1113</sup>

Startups are the new employment...

Angel investors...?

Venture capital is long-tail investing, a high failure rate, with most of the profit coming from a few huge successes, it's a *hits business*...

Open source development may be funded by via individuals' private income...

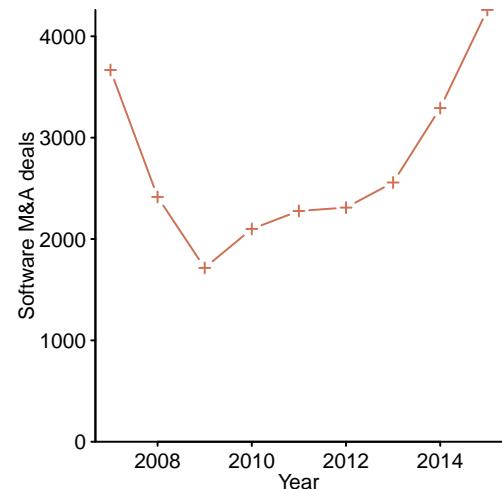


Figure 4.6: Reported number of worldwide software industry mergers and acquisitions (M&A), per year. Data from Solganick.<sup>1467</sup> code

## 4.1.2 Hardware

Computer hardware is the terrain on which software systems live.<sup>ii</sup> The performance and storage capacity limitations of early computers created a software culture that eulogized the use of clever tricks to minimize the amount of code and/or storage required to solve a problem.

The functionality supported by a software system is constrained by the performance and capacity constraints of customer computing platforms. The number of potential customers is likely to increase as the cost of the necessary computing hardware decreases; see Figure 1.1 and Figure 13.11 for illustrations of the cost of operations, and Figure 1.2 for the cost of storage.

Figure 4.7 shows what were considered to be typical memory capacities and costs, for 167 computer systems available from the major vendors, between 1970 and 1978; lines are fitted regression models for 1971, 1974 and 1977; also see [ecosystems/CompWorld85.R](#). The memory capacity of top of the range computers was measured in megabytes, while for small business machines it was measured in kilobytes.

The general classification of computer systems<sup>134</sup> into mainframes, minicomputers<sup>133</sup> and microcomputers was primarily marketing driven,<sup>iii</sup> with each platform class occupying successively lower price points, targeting different kinds of customers (e.g., mainframes large businesses, minicomputers technical and engineering companies, and micros individuals). Supercomputing<sup>135</sup> (i.e., the fastest computers of the day, often used for scientific and engineering calculations) is an example of a significant niche hardware ecosystem that has always existed; embedded systems (where the computing aspect may be invisible to users; see Figure 4.9 and Figure 4.10)...

Figure 4.8 shows performance (in MIPS) against price for 106 computers, from different markets, in 1981. Microcomputer vendors<sup>917</sup> were able to provide solutions that had a much larger market than larger classes of computers, and the high volume market provided the incentives for companies (in particular Intel) to invest in faster processors; see Figure 4.9. Microprocessors eventually achieved comparable processor performance and memory capacity (to more expensive classes of computers), at a lower cost; the lower cost hardware increased sales, which motivated companies to write applications for a wider customer base, which motivated the sale of computers.

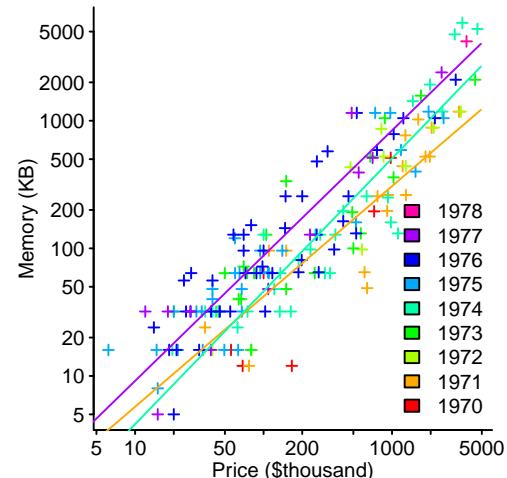


Figure 4.7: Typical memory capacity against cost of 167 different computer systems from 1970 to 1978; fitted regression lines are for 1971, 1974 and 1977. Data from Cale.<sup>240</sup> code

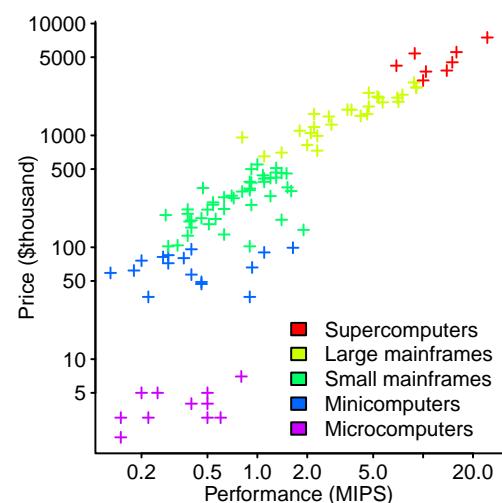


Figure 4.8: Performance, in MIPS, against price of 106 computer systems available in 1981. Data from Ein-Dor.<sup>436</sup> code

<sup>ii</sup> The division between hardware and software can be very fuzzy; for instance, the hardware for Intel's Software Guard Extensions (SGX) instructions consists of software micro operations performed by lower level hardware.<sup>335</sup>

<sup>iii</sup> The general characteristics of central processors and subsystems are very similar, and have followed similar evolutionary paths because they are solving the same technical problems.

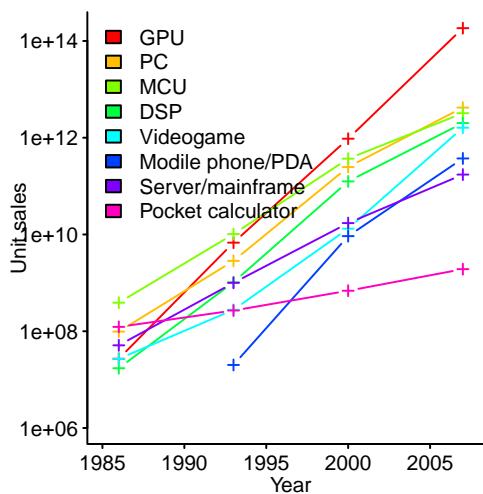


Figure 4.9: Total sales of various kinds of processors.  
Data from Hilbert et al.<sup>698</sup> [code](#)

Manufacturers of computing systems used to regularly introduce new product ranges containing cpus<sup>119</sup> that were incompatible with their existing product ranges. Even IBM, known for the compatibility of its 360/370 family of computers (first delivered in 1965, the IBM 360 was the first backward compatible computer family, that is successive generations could run software that ran on earlier machines), continued to regularly introduced new systems based on cpus that were incompatible with existing systems.

What was the impact on software engineering ecosystems, of businesses having to regularly provide functionality on new computing platforms? One benefit for developers was an expanding job market, where they were always in demand. Having to constantly adapt or reimplement software systems makes it difficult for a substantial body of code and/or practices to become established... Did this regular turnover prevent the premature establishment of practices, or did it simply delay what was always likely to happen?...

For its x86 family of processors, Intel made upwards compatibility a requirement<sup>iv</sup>. An apparently insatiable market demand for faster processors, and large sales volume, created the incentive to continually make significant investments in building faster processors; see Figure 4.9. The x86 family steadily increased market share,<sup>1498</sup> to eventually become dominant in the non-embedded computing market; one-by-one manufacturers of other processors ceased trading, and their computers have become museum pieces.

The market dominance of IBM hardware and associated software (50 to 70% of this market during 1969-1985;<sup>414</sup> see Figure 1.5) is something that developers now learn through reading about the history of computing. However, the antitrust cases against IBM continue to influence how regulators think about how to deal with monopolies in the computer industry, and on how very large companies structure their businesses.<sup>1219</sup>

While the practices and techniques used during one hardware era (e.g., mainframes, mini-computers, microcomputers, the internet) might not be appropriate in a later era, they have left their mark on software culture in the form of language standards written to handle more diverse hardware than exists today<sup>785</sup>... Also, each hardware generation has initially been populated by developers who are relatively new to computing, and thus unfamiliar with what has gone before and so often reinvented things...

Changes in the relative performance of hardware components impacts the characteristics of systems designed for maximum performance, which in turn impacts software design choices, which may take many years to catch up. data on sort algorithms that are cache dependent...

Embedded systems is a software ecosystem that has not attracted as much attention from the software engineering research community as desktop systems. Figure 4.10 shows that in terms of microprocessor sales volume the embedded systems market is significantly larger than what might be called the desktop computer market. mobile market...

The energy consumed by computing devices is an important factor in some markets, from laptop and mobile phone battery usage,<sup>99</sup> compute infrastructure within a building<sup>829</sup> to the design of Warehouse-scale systems<sup>107</sup> (there are large energy efficiency gains to be had running software in the Cloud<sup>1025</sup>). The world-wide energy consumed by computing systems is...

The larger the computing system, the more power it consumes [benchmark/brl/brl.R](#).

The variation in power consumption between supposedly identical components can have a performance impact (i.e., devices reducing their clock rate to keep temperature within limits; see Section 13.3.2.1)...

The semiconductor industry created an organization<sup>1382</sup> whose role is to publish agreed technological roadmaps, outlining the characteristics of devices expected to be available in the coming years. The purpose of these roadmaps is to provide planning information, based on improvements that semiconductor manufacturers are working towards.

Figure 4.11 shows how wafer production revenue at TSMC (the world's largest independent semiconductor foundry) has migrated to smaller process technologies over time, and how demand has shifted across major market segments...

<sup>iv</sup> To the extent of continuing to replicate faults present in earlier processors.<sup>?</sup>

## 4.2 Evolution

The major components of software ecosystems have been rapidly evolving since digital computing began, e.g., hardware capability, customers, vendors and development practices. Clients paying for software systems would like a quiet life, but have to work to maintain their position on the Red Queen treadmill, from fear that competitors will entice away their customers. In the turmoil generated by change, people have more opportunity for change, than if they worked in more mature industries.

Left untouched, software remains unchanged from the day it starts life; use does not cause it to wear out or break. However, the world in which software operates does not remain unchanged, and it is this changing world that reduces the utility of unchanged software. Software systems that have had minimal adaptation, to a substantially changed world,<sup>162</sup> are sometimes known as *legacy systems*.

For instance, overtime companies introduce new processors, and stop supporting those with low sales volume. Compiler vendors can respond by adding support for new processors, and reduce support costs by ceasing to support those processors that are no longer of interest to customers. Figure 4.12 shows the survival curve for processor support in GCC, since 1999, and non-processor specific options.

Software evolves when existing software is modified, or when new software is written. Evolution requires people with the capacity to drive the changes, e.g., ability to make the changes, and/or fund others to do the work.

Incentives for investing to adapt software, to a changed world, include:

- continuing to make money, from existing customers, through updates of an existing product. Updates may not fill any significant new customer need, but some markets have been trained, over decades of marketing, to believe that the latest version is always better than previous versions,
- continuing to make sales of the existing product requires updates to match features available in competing products,
- a potential new market opens up, and modifying an existing product is the most cost effective way of entering this market, e.g., the creation of a new processor creates an opportunity for a compiler vendor to add support for a new cpu instruction set,
- the cost of adapting existing bespoke software is less than the cost of not changing it, i.e., the job the software did, before the world changed, still has to be done,
- software developers having a desire, and the time, to change existing code (the pleasure obtained from doing something interesting is a significant motivation for some of those involved in the production of software systems),

Incentives for investing in the creation of new software include:

- needing to solve a problem, and bespoke software is believed to provide the more cost effective solution,
- profit from selling a product, when not having the option of modifying existing products,
- a growing market share of new hardware platforms; see Figure 4.5...
- a desire to create something. This is a common human desire, and in the case of researchers in academia a requirement for anybody wanting to acquire or enhance an academic reputation.<sup>728</sup> Open source developers...

A product ecosystem sometimes experiences a period of rapid change; exponential improvement in product performance is not computer specific. Figure 4.13 shows the increase in the maximum speed of human vehicles on the surface of the Earth, and in the air, over time.

The evolution of a product may even stop, and restart sometime later. For instance, Microsoft stopped working on their web browser, Internet Explorer, reassigned and laid off the development staff;<sup>7</sup> once this product started to lose significant market share, product development was restarted.

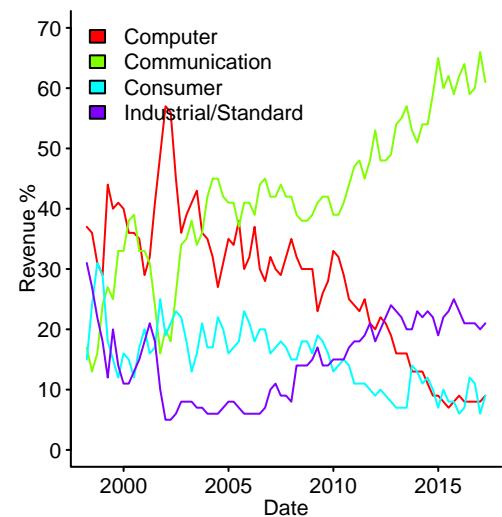
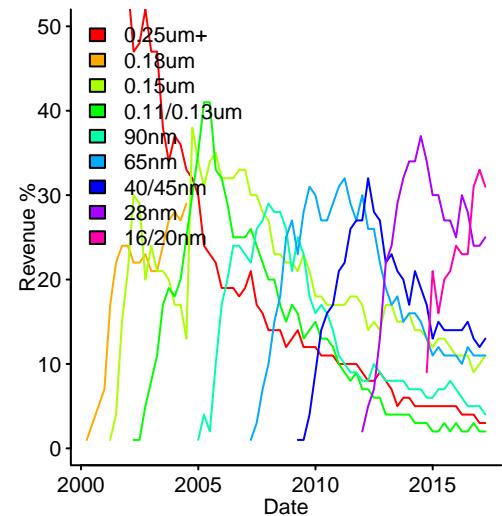


Figure 4.11: TSMC revenue from wafer production, as a percentage of total revenue, at various line widths. Data from TSMC.<sup>1568</sup> [code](#)

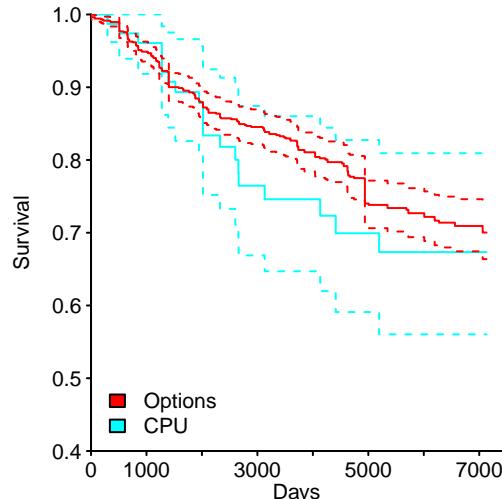


Figure 4.12: Survival curve for GCC's support for distinct cpus and non-processor specific compile-time options. Data extracted from gcc website.<sup>552</sup> [code](#)

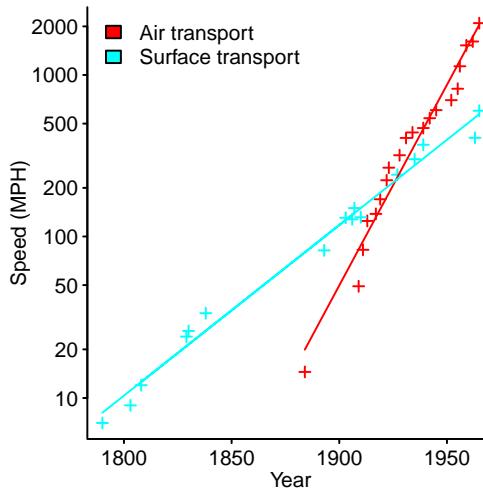


Figure 4.13: Maximum speed achieved by vehicles over the surface of the Earth, and in the air, over time. Data from Lienhard.<sup>961</sup> [code](#)

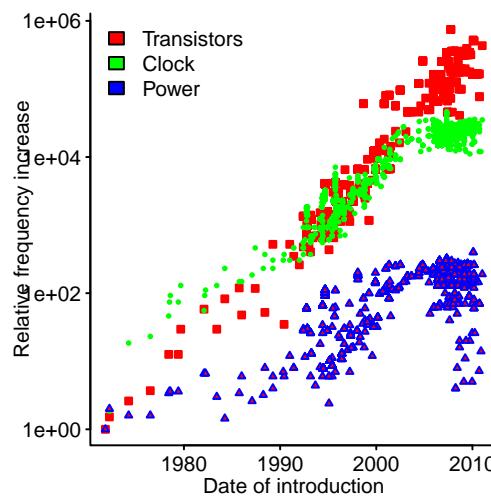


Figure 4.14: Number of transistors, frequency and SPEC performance of cpus when first launched. Data from Danowitz et al.<sup>356</sup> [code](#)

Ecosystem evolution is path dependent.<sup>1493</sup> Things are the way they are today because past decisions selected particular paths to follow for particular reasons, e.g., the QWERTY keyboard layout was designed to reduce jamming in the early mechanical typewriters,<sup>358</sup> not optimizing typist performance (which requires a different layout<sup>294</sup>). The evolutionary paths followed in different locations can be different (e.g., evolution of religious<sup>v</sup> beliefs<sup>1681</sup>).

Some evolutionary paths eventually reach a dead-end. For instance, NEC's PC-98 series of computers, first launched in 1979, achieved sufficient market share in Japan to compete with IBM compatible PCs, until the late 1990s (despite not being compatible with the effective standard computing platform outside of Japan).<sup>1653</sup>

Rapid hardware evolution appears to be coming to an end: processor performance increases having an 18-month cycle is now part of history. Figure 4.14 shows that while the number of transistors in a device has continued to increase, clock frequency has plateaued (the thermal energy generated by running at a higher frequency cannot be extracted fast enough to prevent devices destroying themselves).

When an ecosystem is evolving, rather than being in a steady state, analysis of measurements made at a particular point in time can produce a misleading picture. For instance, a snapshot of software systems currently being maintained may find that the majority have a maintenance/development cost ratio greater than five; however, the data suffers from survivorship bias, the actual ratio is closer to 0.8 (see Figure 4.43 and [ecosystems/maint-dev-ratio.R](#)). With many software ecosystems still in their growth phase, the rate of evolution may mean that patterns found in measurement data change equally quickly.

## 4.2.1 Diversity

Diversity<sup>vi</sup> is important in biological ecology<sup>271</sup> because members of a habitat feed off each other (directly by eating, or indirectly by extracting nutrients from waste products). Software is not its own self-sustaining food web, its energy comes from the people willing to invest in it, and it feeds people who choose to use it. The extent to which ecological diversity metrics used in biology are applicable to software, if at all, is unknown.

If software diversity is defined as variations on a theme, then what motivates these variations and what are the themes?

Software is created by people, and variations between people will produce variations in the software they write; other source of variation include funding, customer requirements, and path dependencies present in the local ecosystem.

Themes include: the software included as part of operating system distributions (e.g., a Linux distribution), the functionality supported by applications providing the same basic need (e.g., text editing), the functionality supported by successive releases of an application, by customised versions of the same release of an application, and the source code used to implement an algorithm (see Figure 9.13).

A study by Keil, Bennett, Bourgeois, Garcá-Peña, MacDonald, Meyer, Ramirez and Yguel<sup>831</sup> investigated the packages included in Debian distributions. Figure 4.15 shows a phylogenetic tree of 56 Debian derived distributions, based on the presence/absence of 50,708 packages in each distribution.

A software system is *forked* when the applicable files are duplicated, and work progresses on this duplicate as a separate entity from the original. The intent may be to later merge any changes into the original, to continue development independently of the original, some combination, or other possibilities (e.g., a company buying a license to adapt software for its internal use).

The *Fork* button on the main page of every project on GitHub is intended as a mechanism for developers, who need not be known to those involved in a project, to easily copy a project from which to learn, and perhaps make changes; possibly submitting any changes back to the original project, a process known as *fork and pull*. As of October 2013, there were 2,090,423 forks of the 2,253,893 non-forked repositories on GitHub.<sup>774</sup>

A study by Robles and González-Barahona,<sup>1340</sup> in 2011, attempted to identify all known significant forks; they identified 220 forked projects, based on a search of Wikipedia articles,

<sup>v</sup> There are over 33,830 denominations, with 150 having more than 1 million followers.<sup>105</sup>

<sup>vi</sup> Number of species and their abundance.

followed by manual checking. Figure 4.16 suggests that after an initial spurt, the number of forks has not been growing at the same rate as the growth of open source projects.

The BSD family of operating systems arose from forking during the early years of their development, and have evolved as separate but closely related projects since the early-mid 1990s. Figure 9.21 suggests how a few developers working on multiple projects communicate bug fixes between them.

A study by Ray<sup>1311</sup> investigated the extent to which code created in one of NetBSD, OpenBSD or FreeBSD was ported to either of the other two versions, over a period of 18 years. Ported code not only originated in the most recently written code, but was taken from versions released many years earlier. Figure 4.17 shows the contribution made by 14 versions of NetBSD (versions are denoted by stepping through the colors of the rainbow) to 31 versions of OpenBSD; the contribution is measured as percentage of lines contained in all the lines changed in a given version.

Products are sometimes customised for specific market segments. Customization might be used to simplify product support, adapt to hardware resource constraints, and market segmentation as a sales tool.

Customization might occur during program start-up, with configuration information being read and used to control access to optional functionality, or at system build time, e.g., optional functionality is selected at compile time, creating a customised program.

Each customized version of a product can experience its own evolutionary pressures,<sup>1217</sup> and customization is a potential source of mistakes.

A study by Rothberg, Dintzner, Ziegler and Lohmann<sup>1354</sup> investigated the number of optional Linux features shared (i.e., supported) by a given number of processor architectures (for versions released between May 2011 and September 2013, during which the number of supported architectures grew from 24 to 30). Table 4.1 shows that the number of features supported by all architecture grew at a faster rate than the number of features supported by just one architecture.

The official Linux kernel distribution does not include all variants that exist in shipped products;<sup>678</sup> while manufacturers may make the source code of their changes publically available, they either do not submit these changes to become part of the mainline distribution, or their submissions are not accepted into the official distribution (it would be a heavy burden for the official Linux kernel distribution to include every modification made by a vendor shipping a modified kernel).

The number of optional features increases as the size of programs increases (see Figure 7.6)...

When a new version of a software system becomes available, there may be little, or no incentive for users to upgrade. As part of customer support, vendors may continue to provide support for previous versions, for some period after the release of a new version.

Version	1	2	3	All-3	All-2	All-1	All
<b>2.6.39</b>	3,989	182	50	2,293	944	1,189	2,617
<b>3.0</b>	3,990	183	53	2,345	968	1,211	2,637
<b>3.1</b>	4,026	184	52	2,440	968	1,155	2,667
<b>3.2</b>	4,028	181	57	1	2,788	512	4,054
<b>3.3</b>	4,077	180	51	1	2,837	512	4,133
<b>3.4</b>	4,087	183	51	1	2,907	520	4,184
<b>3.5</b>	4,129	179	50	2	3,001	520	4,265
<b>3.6</b>	4,158	184	51	2	3,098	527	4,298
<b>3.7</b>	4,139	183	50	1	3,173	539	4,384
<b>3.8</b>	4,148	178	35	3	3,269	548	4,399
<b>3.9</b>	4,269	177	36	3	3,403	581	4,413
<b>3.10</b>	4,280	173	35	3	3,447	577	4,460
<b>3.11</b>	4,270	178	33	2	0	0	8,654

Table 4.1: Number of distinct features, in Linux, shared across (i.e., supported) a given number of architectures (header row), for versions 2.6.39 to 3.11; All denotes all supported architectures (All-1 is one less than All, etc), which is 24 in release 2.6.39, growing to 30 by release 3.9. Data from Rothberg et al.<sup>1354</sup>

The desktop market growth to dominance of Wintel<sup>vii</sup> reduced platform diversity (i.e., the alternatives went out of business), which reduced the incentives for companies to continue

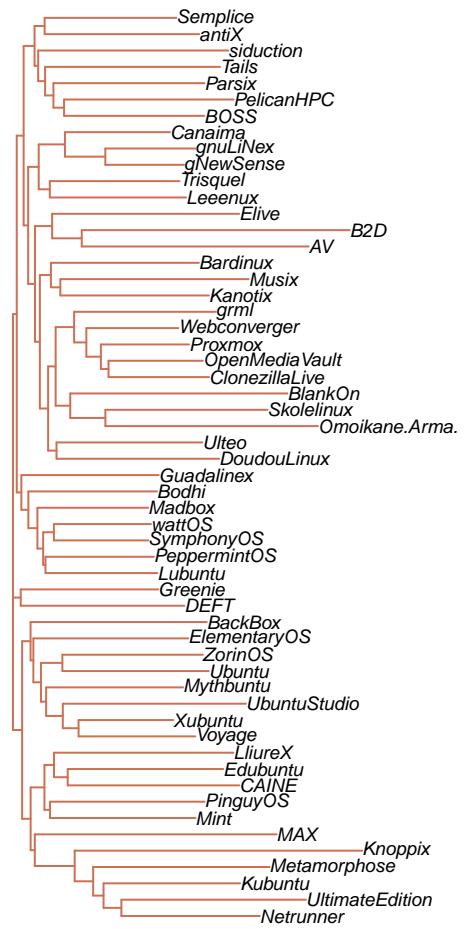


Figure 4.15: Phylogenetic tree of Debian derived distributions, based on which of 50,708 packages are included in each distribution. Data from Keil et al.<sup>831</sup> code

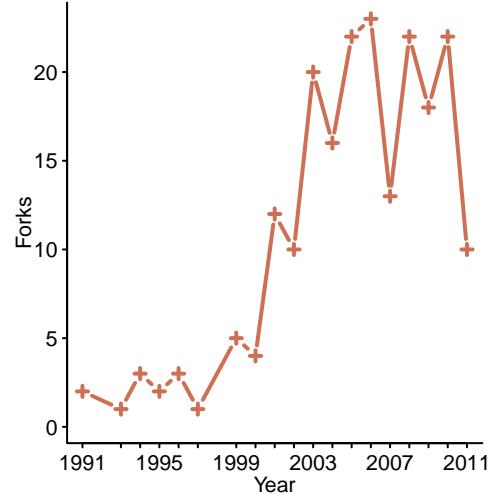


Figure 4.16: Number of forked projects per year, identified using Wikipedia during August 2011. Data from Robles et al.<sup>1340</sup>

<sup>vii</sup> Microsoft Windows coupled with Intel's x86 family of processors.

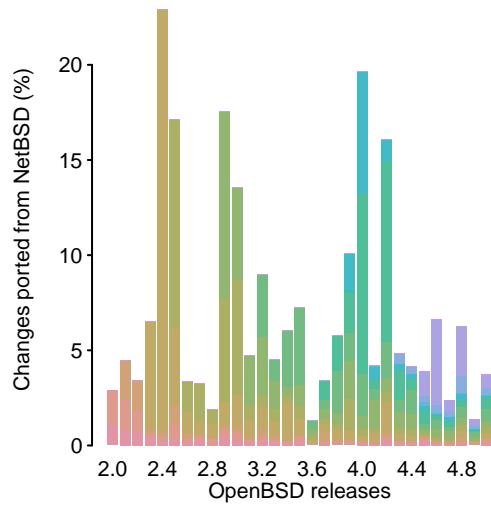


Figure 4.17: Percentage of code ported from NetBSD to various versions of OpenBSD, broken down by version of NetBSD in which it first occurred (denoted by incrementally changing color). Data kindly provided by Ray.<sup>1311</sup>

supporting a diverse range of platforms. In some markets, the primary source of incoming was derived from supporting applications running on the dominant computing platform...

Widely used software systems may support their own ecosystem of third-party add-ons. For instance, Wordpress is written in PHP and third-party add-ons are executed by the version of PHP installed on the server running Wordpress. The authors of these add-ons have to consider the possibility that users are running both an earlier version of Wordpress and an earlier version of PHP.

A study by Ruohonen and Leppänen<sup>1363</sup> investigated the version of PHP available on more than 200K websites (using data from the httrackarchive<sup>730</sup>). Figure 4.18 shows the number of websites running a given version of PHP on the first day of February 2016 and February 2017; the x-axis is ordered by version number of the release (there is some intermingling of dates).

## 4.2.2 Lifespan

The expected lifespan of a software system is of interest to those looking to invest in it. The investment may be vendor related (e.g., designing with an eye to future product sales) or customer related (e.g., integrating use of the software into business workflow). The analysis of data where the variable of interest is measured in terms of *time-to-event*, is known as *survival analysis*; see Section 11.11.

Products expected to have a short lifespan are likely to receive little or no investment (e.g., creating a more robust product, to reduce future maintenance costs), and potential customers may need to be convinced there is a viable migration path to future generations.

In theory, a software system can only said to be dead when one or more of the files needed to run it ceases to exist. In practice, the lifespan of interest to those involved, is the expected period of their involvement with the software system.

In a changing commercial ecosystem, the incentive to maintain and support existing products can disappear. It may be more profitable to create new products without concern for compatibility with current products, current products may not have sold well enough to make further investment worthwhile, or the market itself may shrink to the point where it is not economically viable to continue operating in it.

Hardware wears out, or breaks, and vendors have a revenue stream from replacement sales; software does not wear out, there are no replacement sales. While software does not wear out, it is intertwined with components of an ecosystem, and changes to third-party components that are depended upon can cause programs to malfunction, or even fail to execute. Changes in the outside world can cause unchanged software to stop working as intended.

Manufacturing hardware requires an infrastructure, and specific components will have their own manufacturing requirements. The expected profit from future sales of a device may not make it worthwhile continuing to manufacture, e.g., the sales volume of hard disks is decreasing, as NAND memory capacity performance, and cost per-bit improves.<sup>518</sup>

Some vendors continue to support and maintain products for a period after the next generation of products has been launched, providing an opportunity for customers to decide when to migrate to a later version. data?...

The rate of product improvements can slow down for various reasons, including: technology maturity, or a single vendor comes to dominate the market (i.e., enjoys monopoly-like power). Figure 4.19 illustrates how the working life of jet-engined aircraft (invented in the same decade as computers) has increased over time. Figure 4.20 shows the mean age of installed mainframes at a time when the market was dominated by a single vendor (IBM), who decided product pricing and lifespan; the vendor could influence product lifespan to maximise their revenue.

Organizations and individuals create and support their own Linux distribution, often derived from one of the major distributions (e.g., Ubuntu was originally derived from Debian). Lundqvist and Rodic<sup>987</sup> recorded the life and death of these distributions; Figure 4.21 shows the survival curve, stratified by the parent distribution.

The market share of the latest version of a software system typically grows to a peak, before declining as newer versions are released. Villard<sup>1613</sup> tracked the Android version usage over

time. Figure 4.22 shows the percentage share of the Android market held by various releases, based on days since launch of each release.

A study by Tamai and Torimitsu<sup>1534</sup> investigated the lifespan of 95 software systems (which appear to be mostly in-house systems). Figure 4.23 shows (red) the number of systems terminated after a given number of days, along with a fitted regression model of the form:  $systems = ae^{b \times years}$  (with  $b = -0.14$  for the mainframe software and  $b = -0.24$  for Google's SaaS).

A website setup and maintained by Ogden<sup>1181</sup> records Google applications, services and hardware that have been terminated. Figure 4.23 shows (blue/green) the number of that have been terminated after a given number of days, along with a fitted regression model.

System half-life for the 1991 Japanese corporate mainframe data is almost 5-years; for the Google SaaS it is almost 2.9-years. Does the Japanese data represent a conservative upper bound, and the Google data a lower bound for a relatively new, large company finding out which of its original products are worth keeping?

Communities of people have a lifespan. When the energy for the community comes directly from commercially activity, the community will continue to exist for as long as there are people willing to work for the money available. When the energy comes directly from those active in the community, community existence is dependent on the motivation and beliefs of its members...

A study by Dunbar and Sosis<sup>417</sup> investigated human community sizes and lifetime. Figure 4.24 shows the number of founding members of 53, 19th century secular and religious utopian communities, along with the number of year they continued to exist...

### 4.2.3 Entering a market

Markets for particular classes of products (e.g., electric shavers, jet engines and video cassette recorders) and services evolve, with firms entering and existing. Manufacturing markets have been found to broadly go through five stages, from birth to maturity.<sup>10</sup>

The invention of the Personal Computer triggered the creation of new markets, including people starting manufacturing companies to seek their fortune selling lower cost computers. Figure 4.25 shows how the growth and consolidation of PC manufacturers followed a similar trend to the companies started to manufacture automobiles (i.e., hundreds of companies started, but only a few survived).

Vendors want to control customer evolution, at least to the extent to which it involves product purchasing decisions. Evolution might be controlled by erecting barriers (the term *moat* is also used) to either make it unprofitably for other companies to enter the market, or to increase the cost for customers to switch suppliers.

- *economies of scale* (a supply side effect, a traditional barrier employed by hardware manufacturers): producing in large volumes allows firms to spread their fixed costs over more units, improving efficiency (requires that production be scalable in a way that allows existing facilities to produce more). Competing against a firm producing in volume requires a large capital investment to enter the market, otherwise the new entrant is at a cost disadvantage.

With effectively zero replication costs, once created software systems do not require a large capital investment. A large percentage of the cost of software production is spent on people, who provide few opportunities for economies of scale (there may even be diseconomies of scale, e.g., communication overhead increases with head count),

- *network effects*<sup>61</sup> (a demand-side effect): are created when customers' willingness to buy from a supplier increases with the number of existing customers. Competing against an established firm selling into a market with network effects requires large a capital investment by a company entering the market, to offer incentives so that a significant percentage of customers will switch suppliers...
- *switching costs*:<sup>473</sup> a switching cost is an investment specific to the current supplier that must be duplicated for a new supplier, e.g., retraining staff and changes to procedures, e.g., the UK government paid £37.6 million transition costs to the winning bidder of the ASPIRE contract, with £5.7 million paid to the previous contract holder to facilitate changeover.<sup>318</sup>

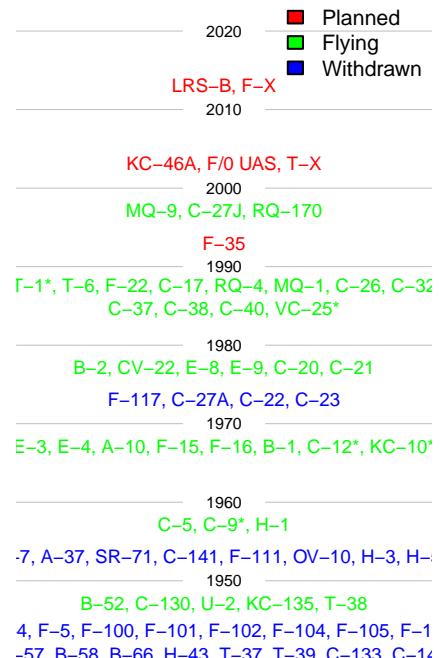


Figure 4.19: Decade in which newly designed US Air Force aircraft first flew, with colors indicating current operational status. Data from Echbeth et al.<sup>428</sup> code

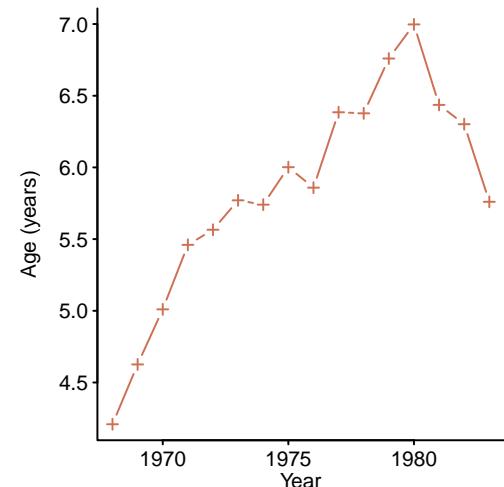


Figure 4.20: Mean age of installed mainframe computers, 1968-1983. Data from Greenstein.<sup>623</sup> code

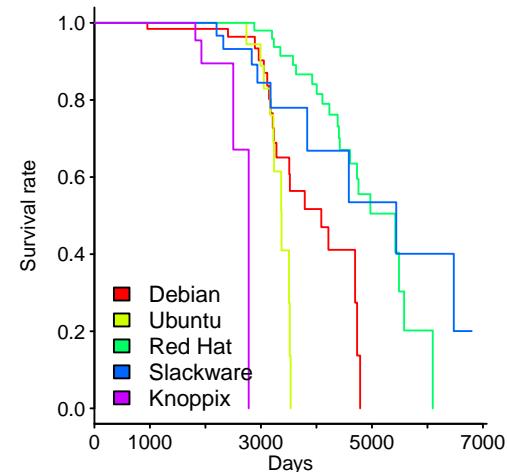


Figure 4.21: Survival curve of Linux distributions derived from five widely-used parent distributions (identified in legend). Data from Lundqvist et al.<sup>987</sup> code

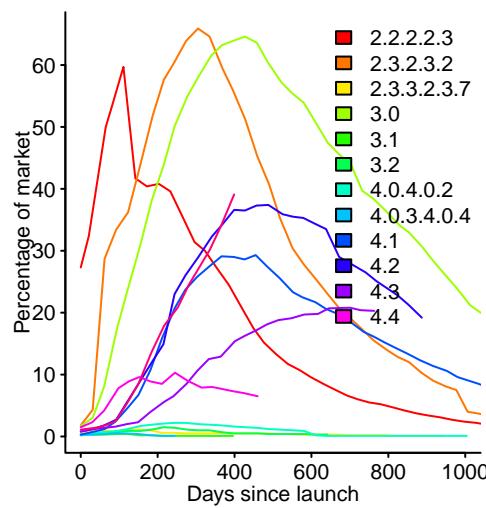


Figure 4.22: Percentage share of Android market, of a given release, by days since its launch. Data from Villard.<sup>1613</sup> [code](#)

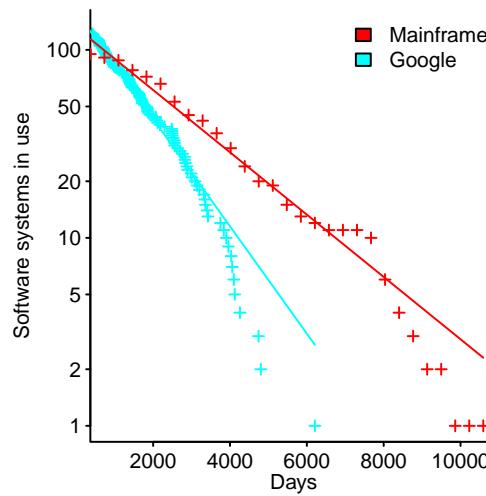


Figure 4.23: Number of software systems surviving for a given number of days and fitted regression models: Japanese mainframe software (red), Google software-as-a-service (blue). Data from: mainframe Tamai,<sup>1534</sup> Google's SaaS Ogden.<sup>1181</sup> [code](#)

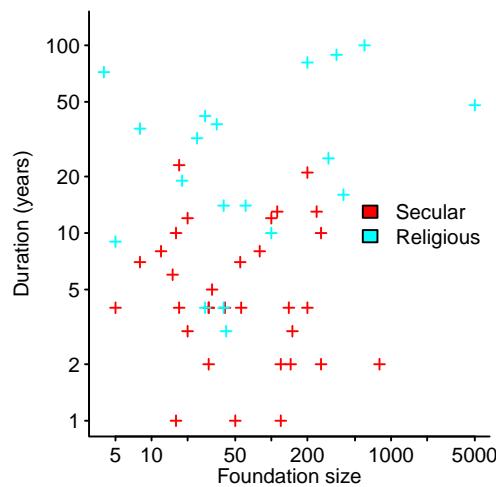


Figure 4.24: Size at foundation and lifetime of 53 secular and religious 19th century American utopian communities. Data from Dunbar et al.<sup>417</sup> [code](#)

Information asymmetry can create switching costs by deterring competition; to encourage competition in bidding on the replacement ASPIRE contract, the incumbent had inside knowledge and was perceived to be strong, the UK government contributed £8.6 million towards bidders' costs.<sup>318</sup>

A vendor with a profitable customer base, protected by products with high switching costs, is incentivised to prioritise the requirements of customers inside their walled garden. Income from existing customers sometimes causes vendors to ignore new developments that eventually lead to major market shifts,<sup>474</sup> that leave the vendor supporting a marooned customer based.

Ho hum... the data...

The competitive forces faced by a company include:<sup>1266</sup> rivalry between existing competitors, bargaining power of suppliers, bargaining power of buyers, possibility of new entrants, and possibility product being substituted by alternative products or services...

The Red Queen model,<sup>103</sup> data promised'ish...

## 4.3 Population dynamics

Population dynamics<sup>1043, 1168</sup> is a fundamental component of ecosystems.

Connections between members of an ecosystem are not always random. When network connections are driven by a non-random process, surprising properties can emerge, e.g., the *friends paradox*, where your friends have more friends, on average, than you do,<sup>482</sup> and the *majority illusion*, where views about a complete network are inferred from the views of local connections.<sup>945</sup> In general, if there is a positive correlation, for some characteristic, between connected nodes, a node's characteristic will be less than the average of the characteristic for the nodes connected to it.<sup>449</sup>

The choices made by individual members of an ecosystem can have a significant impact on population dynamics, which in turn can influence subsequent individual choices. Two techniques used to model population dynamics are: mathematics and simulation (covered in Section 12.5).

The mathematical approach is often based on the use of differential equations: the behavior of the important variables are specified in one or more equations, which may be solved analytically or numerically.

An example is the evolution of the population of two distinct entities within a self-contained ecosystem. If, say, entities *A* and *B* have fitness *a* and *b* respectively, both have growth rate *c*, and an average fitness of  $\phi$ , then the differential equation describing the evolution of their population size, *x* and *y*, over time is given by:<sup>1168</sup>

$$\dot{x} = ax^c - \phi x$$

$$\dot{y} = by^c - \phi y$$

Solving these equations shows that, when  $c < 1$ , both *A* and *B* can coexist, when  $c = 1$ , the entity with the higher fitness can invade and dominate an ecosystem (i.e., lower fitness eventually dies out), but when  $c > 1$ , an entity with high fitness cannot successfully invade an occupied ecosystem (i.e., greater fitness is not enough to displace an incumbent).

The mathematics approach has the advantage that, if the equations can be solved, the behavior of a system can be read from the equations that describe it (while simulations provide a collection of answers to specific initial conditions). The big disadvantage of the mathematical approach is that it may not be possible to solve the equations describing a real world problem.

The advantage of simulations is that they can handle most real world problems. The disadvantages of simulations include: difficulty of exploring the sensitivity of the results to changes in model parameters, difficulty of communicating the results to others, and computational cost; for instance, if  $10^4$  combinations of different model parameter values are needed to cover the possible behaviors in sufficient detail, with  $10^3$  simulations for each combination (needed to reliably estimate the mean outcome), then  $10^7$  simulation runs are needed, which at 1 second each is 116 cpu days.

When a product ecosystem experiences network effects, it is in vendors' interest to create what is known as a *virtuous circle*; where, encouraging third-party developers to sell their

products within the ecosystem attracts more customers, which in turn attracts more developers, and so on.

Given two new technologies, say A and B, competing for customers in an existing market, what are the conditions under which one technology comes to dominate the market<sup>60</sup>?

Assume that at some random time, a customer has to make a decision to replace their existing technology, and there are two kinds of customer: R-agents perceive a greater benefit in using technology A (i.e.,  $a_R > b_R$ ), and S-agents perceive a greater benefit in using technology B (i.e.,  $a_S < b_S$ ); both technologies are subject to network effects, i.e., having other people using the same technology provides a benefit to everybody else using it.

Figure 4.26 illustrates the impact of the difference in customers of two products (y-axis), with time along the x-axis; red-line is an example difference for the number of customers using products A and B. Between the blue/green lines, R and S-agents perceive a difference in product benefits; once the difference in the number of customers of each product crosses a line, both agents perceive one product to have the greater benefit.

Table 4.2 shows the total benefit available to each kind of customer, from adopting one of the technologies;  $n_A$  and  $n_B$  are the number of users of A and B at the time a customer makes a decision,  $r$  and  $s$  are the benefits accrued to the respective agents from existing users (there are increasing returns when:  $r > 0$  and  $s > 0$ , decreasing returns when:  $r < 0$  and  $s < 0$ , and no existing user effect when:  $r = 0$  and  $s = 0$ ).

	Technology A	Technology B
R-agent	$a_R + rn_A$	$b_R + rn_B$
S-agent	$a_S + sn_A$	$b_S + sn_B$

Table 4.2: Returns from choosing A or B, given previous technology adoptions by others. From Arthur.<sup>61</sup>

For increasing returns, lock-in of technology A occurs<sup>61</sup> (i.e., it provides the greater benefit for all future customers) when:  $n_A(t) - n_B(t) > \frac{b_S - a_S}{s}$ , where:  $n_A(t)$  and  $n_B(t)$  are the time dependent values of  $n$ .

The condition for lock-in of technology B is:  $n_B(t) - n_A(t) > \frac{a_R - b_R}{r}$

Starting from a market with both technologies having the same number of customers, the probability that technology A eventually dominates is:  $\frac{s(a_R - b_R)}{s(a_R - b_R) + r(b_S - a_S)}$  and technology B dominates with probability:  $\frac{r(b_S - a_S)}{s(a_R - b_R) + r(b_S - a_S)}$

With decreasing returns, both technologies can coexist.

Governments have passed laws intended to ensure that the competitive process works as intended within commercially important ecosystems (in the US this is known as *antitrust law*, while elsewhere the term *competition law* is often used). In the US antitrust legal thinking<sup>839</sup> in the 1960s was based on a market structure-based understanding of competition (i.e., courts blocked company mergers that they thought would lead to anticompetitive market structures). This shifted in the 1980s, with competition assessment based on the short-term interests of consumers (i.e., low consumer prices), not based on producers, or the health of the market as a whole.

The legal decisions and rules around ensuring that the competitive process operates in the commercial market for information are new and evolving.<sup>1219</sup>

In the UK and US it is not illegal for a company to have monopoly power within a market. It is abuse of a dominant market position that gets the attention of authorities; governments sometimes ask the courts to block a merger because they believe it would significantly reduce competition.<sup>1399</sup>

### 4.3.1 Growth processes

Population dynamics are sometimes modeled using a one-step at a time growth algorithm. A commonly encountered example is preferential attachment, which is discussed in Section 8.3.1.

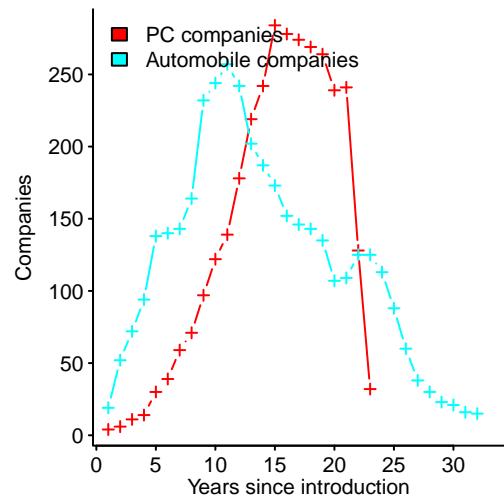


Figure 4.25: Number of US companies manufacturing automobiles and PCs, over the first 30-years of each industry. Data extracted from Mazzucato.<sup>1032</sup> code

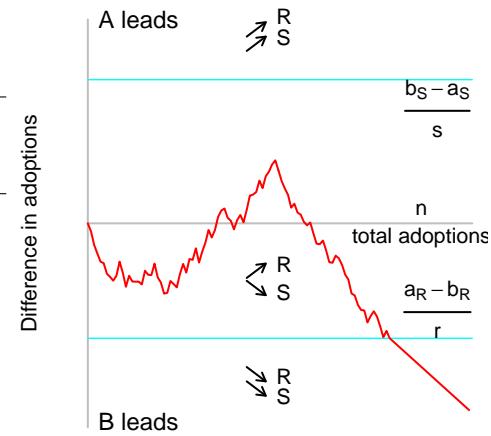


Figure 4.26: Example showing difference in number of customers using two products. code

The percentage breakdown of subpopulations within a population, can be path dependent. Consider the population of an urn containing one red and one black ball. The growth process (known as the *Polya urn process*) iterates the following sequence: a ball is randomly drawn from the urn, this ball, along with another ball of the same color is placed back in the urn. After a many iterations, what is the percentage of red and black balls in the urn?

Polya proved that the percentage of red and black balls always converges to some, uniformly distributed, random value. Each converged value is the outcome of the particular sequence of (random) draws that occurred early in the iteration process; any converged percentage between zero and 100 can occur. Convergence also occurs when the process starts with an urn containing more than two balls, with each ball having a different color; see [ecosystems/Polya-urn.R](#).

The Polya urn process has been extended to include a variety of starting conditions (e.g., an urn containing multiple balls of the same color) and replacement rules (e.g., adding multiple balls of the same color, or balls having the other color). General techniques for calculating exact solutions to problems involving balls of two colors are available.<sup>503</sup>

spin-off companies...

growth in the number of documents and spreadsheets created using Microsoft Office or OpenOffice<sup>1352</sup>...

Diffusion processes... The diffusion of software systems<sup>331</sup> across the world... The Bass model is discussed in Section 3.6.3.

Over time various techniques have been developed to make it more difficult for viruses to hijack programs; it takes time for effective techniques to be discovered, implemented and then used by developers. Figure 4.27 shows the growth in the number of programs in the Ubuntu AMD64 distribution that are hardened with a given security technique (Total ELF is the total number of programs).

Software change is driven by a mixture of business and technical factors. A study by Branco, Xiong, Czarnecki, Küster and Völzer<sup>199</sup> analysed over 1,000 change requests in 70 business process models of the Bank of Northeast of Brazil. Figure 4.28 shows the distribution of the 388 maintenance requests made during the first three years of the Customer Registration project. Over longer time-scales the rate of evolution of some systems has a cyclic pattern.<sup>1701</sup>

Some very large systems grow through the integration of independently developed projects.

A study by Bavota, Canfora, Di Penta, Oliveto and Panichella<sup>120</sup> investigated the growth of projects in the Apache ecosystem. Figure 4.29 shows the growth in the number of projects, and coding constructs they contain. The number of dependencies between projects increases as the number of projects increases, along with the number of methods and/or classes (see [ecosystems/icsm2013\\_apache.R](#)).

A later study by the same group<sup>121</sup> analyzed the 147 Java projects in the Apache ecosystem. Figure 4.30 shows, for each pair of projects, the percentage overlap of developers contributing to both projects (during 2013; white none, red 100%); the rows/columns have been reordered to show project pair clusters sharing a higher percentage of developers (see Section 12.4.1).

### 4.3.2 Estimating population size

It may be impossible, or too costly, to observe all members of a population. Depending on the characteristics of the members of a population, it may be possible to estimate of the number of members based on a sample; population estimates might also be derived from a theoretical analysis of the interaction between members.<sup>942</sup>

In some cases it is possible to repurpose existing data. For instance, the U.S. Bureau of Labor Statistics (BLS) publishes national census information,<sup>231</sup> which includes information on citizens' primary occupation. The census occupation codes starting with 100, 101, 102, 104, and 106 have job titles that suggest a direct involvement in software development, with other codes suggesting some involvement. This information provides one means of estimating the number of software developers in the U.S. (one analysis<sup>1254</sup> of the 2016 data estimated between 3.4 million and 4.2 million).

One technique for estimating the number of fish in a lake is to capture fish for a fixed amount of time, count and tag them, before returning the fish to the lake. After allowing the captured

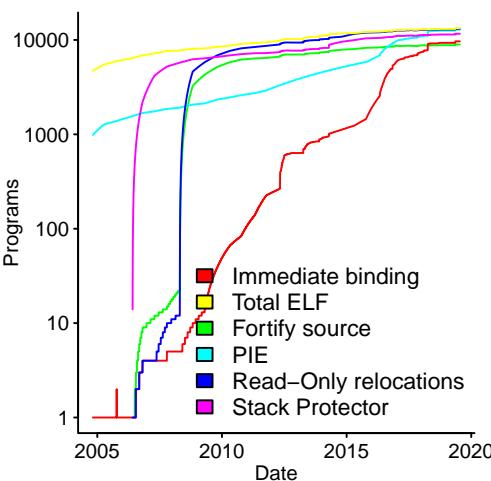


Figure 4.27: Number of programs in the Ubuntu AMD64 distribution shipped using a given security hardening technique (Total ELF is the number of ELF executables). Data from Cook<sup>325</sup> code

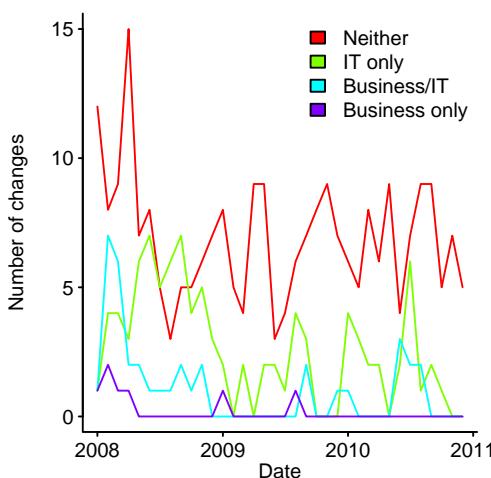


Figure 4.28: Number of process model change requests made in three years of a banking Customer registration project. Data kindly provided by Branco.<sup>199</sup> code

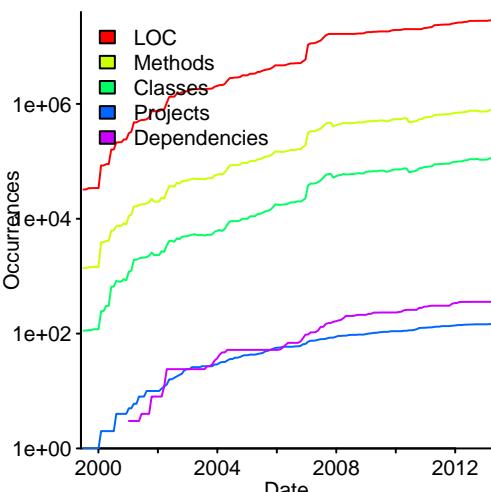


Figure 4.29: Growth in the number of projects within the Apache ecosystem, along with the amount of contained code. Data from Bavota et al.<sup>120</sup> code

fish to disperse, the process is repeated, this time counting the number of tagged fish, and those captured for the first time (i.e., untagged).

The *Chapman estimator* is an unbiased, and more accurate estimate,<sup>1369</sup> than the simpler formula usually derived (i.e.,  $N = \frac{C_1 C_2}{C_{12}}$ ). Assuming that all fish have the same probability of being caught, and the probability of catching a fish is independent of catching any other fish:

$$N = \frac{(C_1 + 1)(C_2 + 1)}{C_{12} + 1} - 1, \text{ where: } N \text{ is the number of fish in the lake, } C_1 \text{ the number of fish caught on the first attempt, } C_2 \text{ the number caught on the second attempt, and } C_{12} \text{ the number caught on both attempts.}$$

Using the Chapman estimator to estimate the number of issues not found by a code review, say, assumes that those involved are equally skilled, invest the same amount of effort in the review process, and all issues are equally likely to be found. When reviewers have varying skills, invest varying amounts of effort, or the likelihood of detecting issues varies, the analysis is more complicated. The Rcapture package supports the analysis of capture-recapture measurements where capture probability varies across items of interest, and those doing the capturing. The VGAM package...

When sampling from a population whose members have been categorized in some way (e.g., by species), two common kinds of sampling data are: *abundance data* which contains the number of individuals within each species in the sample, and *incidence data* giving a yes/no for the presence/absence of each species in the sample. Techniques for estimating population size...

The *Chao1 estimator*<sup>270</sup> gives a lower bound on the number of members in a population, based on a count of each member captured; it assumes that each member of the population has its own probability of capture, that this probability is constant over all captures, and the population is sampled with replacement:

$$S_{est} \geq S_{obs} + \frac{n-1}{n} \frac{f_1^2}{2f_2}$$

where:  $S_{est}$  is the estimated number of unique members,  $S_{obs}$  the observed number of unique members,  $n$  the number of members in the sample,  $f_1$  the number of items captured once, and  $f_2$  the number of members captured twice.

If a population, containing  $N$  members, is sampled without replacement, the unseen member estimate added to  $S_{obs}$  becomes:  $\frac{f_1^2}{\frac{n-1}{n}2f_2 + \frac{q}{1-q}f_1}$ ,<sup>274</sup> where:  $q = \frac{n}{N}$ .

Taking into account members occurring three and four times gives an improved lower bound.<sup>291</sup>

The ChaoSpecies function in the SpadeR package calculates species richness using a variety of models. The SpadeR and iNEXT packages contain functions for estimating and plotting species abundance data.

The number of additional members,  $m_g$ , that need to be sampled to be likely to encounter a given fraction,  $g$ , of all expected unique members in the population is:<sup>272</sup>

$$m_g \approx \frac{nf_1}{2f_2} \log \left[ \frac{f_0}{(1-g)S_{est}} \right]$$

where:  $n$  is the number of members in the current sample and  $f_0 = \frac{f_1^2}{2f_2}$ . For  $g = 1$ , the following relationship needs to be solved for  $m$ :  $2f_1 \left(1 + \frac{m}{n}\right) < e^{\frac{m}{n} \frac{2f_2}{f_1}}$

If  $m$  additional members are sampled, the expected number of unique members encountered is,<sup>1422</sup> assuming  $m < n$  (when  $n \leq m \leq n \log n$ , more complicated analytic estimates are available<sup>1194</sup>):

$$S(n+m) = S_{obs} + f_0 \left[ 1 - \left( 1 - \frac{f_1}{nf_0 + f_1} \right)^m \right]$$

If  $m$  is much less than  $n$ , this equation approximates to:  $S(n+m) \approx S_{obs} + m \frac{f_1}{n}$ .

The formula to calculate the number of unique members shared by two populations is based on the same ideas, and is somewhat involved.<sup>1210</sup>

When capture probabilities vary by time and individual animal<sup>273</sup>...

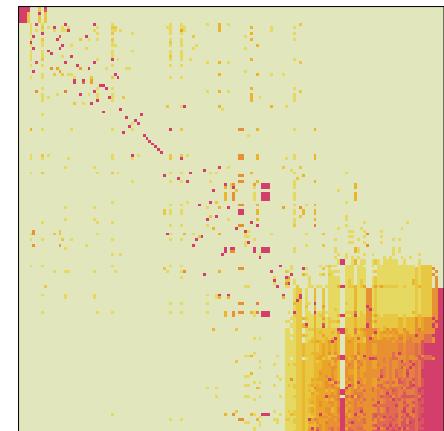


Figure 4.30: Percentage overlap of developers contributing, during 2013, to both of each pair of 147 Apache projects. Data kindly provided by Panichella.<sup>121</sup> [code](#)

## 4.4 Organizations

Organizations contain ecosystems of people who pay for, use and create software systems.

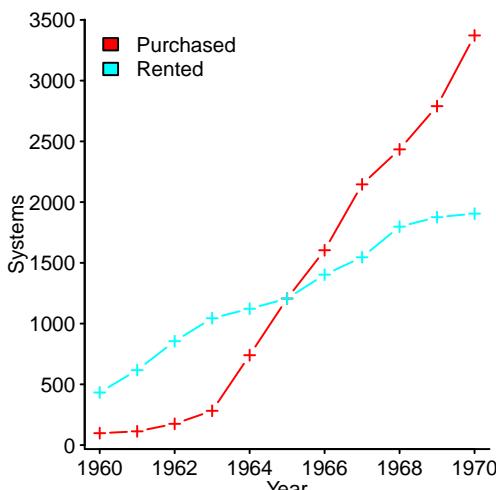


Figure 4.31: Total computer systems purchased and rented by the US Federal Government in the respective fiscal years ending June 30. Data from US Government General Accounting Office.<sup>319</sup> [code](#)

### 4.4.1 Customers

Given that customers supply the energy that drives software ecosystems, the customers supplying the greatest amount of energy are likely to have the greatest influence on software engineering culture and practices. Culture is built up over time, and is path dependent, i.e., practices established in response to events early in the history of a software ecosystem can continue, may continue to be followed long after they have ceased to provide any benefits.

The military were the first customers for computers, and financed the production of one-off systems.<sup>504</sup> The first commercial computer, the Univac I, was introduced in 1951, and 46 were sold; the IBM 1401,<sup>547</sup> introduced in 1960, was the first computer to exceed one thousand installations, with an estimate 8,300 systems in use in July 1965.<sup>998</sup> During the 1950s a steady stream of new computers were introduced, with many customers initially preferring to rent (management feared equipment obsolescence in a rapidly changing market, and had little desire to be responsible for maintenance<sup>901</sup>) and vendors nearly always preferring to rent (a license agreement could restrict customers' ability to connect cheaper peripherals to equipment of they do not own) preferred to rent. Figure 4.31 shows the shift from rental to purchase of computers by the US Federal Government.

The U.S. Government was the largest customer for computing equipment during the 1950s and 60s, and enacted laws to require vendors to supply equipment that conformed to various specified standards, e.g., the Brooks Act<sup>1558</sup> in 1965. The intent of these standards was to reduce costs, to the government, by limiting vendors ability to make use of proprietary interfaces to restrict competition.

The customers for these very expensive early computers were large organizations who had to regularly perform numeric calculations on a large scale, e.g., payroll (these organizations employed thousands of people, each requiring a unique weekly or monthly payroll calculation<sup>622</sup>), making forecasts and calculating engineering tables.

Large organizations had problems that were large enough to warrant the high cost of buying and operating a computer.<sup>1415</sup> A computer services industry<sup>1689</sup> quickly grew (during 1957, in the US, 32 of the 71 systems offered were from IBM<sup>997</sup>), providing access to computers in central locations (often accessed via dial-in phone lines), with processing time rented by the hour.<sup>67</sup> Computing as a utility service for general use, like electricity, looked to become a viable business model.<sup>1407</sup> Figure 4.32 shows, for the US, service business revenue in comparison to revenue from system sales.

The introduction of microcomputers decimated the computer services business;<sup>247</sup> the cost of buying a microcomputer could be less than the monthly rental to a service bureau.

Unless sold embedded in a hardware device (e.g., IoT and military equipment), the characteristics of the computers on which a software system has to run is a major consideration.

There may be very specific hardware requirements (e.g., games consoles and *super computers*; see [Rlang/Top500.R](#)), or software requirements (e.g., the operating system running on a desktop computer; Section 4.5.3 discusses installed libraries). For some problems the techniques involved in their solution are sufficiently different that bespoke hardware<sup>1417</sup> is cost effective.

How much do customer ecosystems spend on software?

Figure 4.33 shows the total yearly amount spent, by the UK's 21 industry sectors, on their own software (which was reported by companies as fixed-assets; money may have been spent on software development that for accounting purposes was not treated as a fixed-asset).

What problems are customers interesting in solving?

scientific/engineering computing: the substantial number of calculations<sup>7</sup> required to answer some economically important problems (e.g., weather prediction) has created a niche market for compute intensive systems...

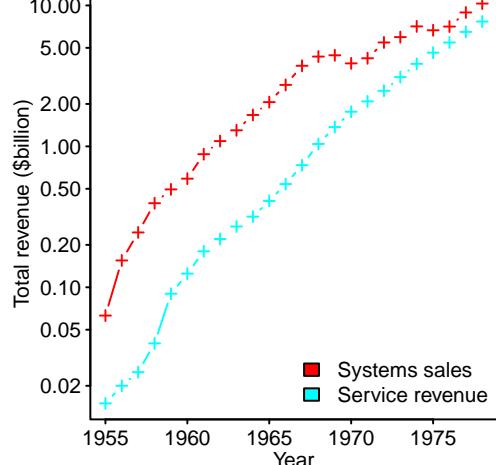


Figure 4.32: Total U.S. revenue from sale of computer systems and data processing service industry revenue. Data from Phister<sup>1245</sup> table II.1.20 and II.1.26. [code](#)

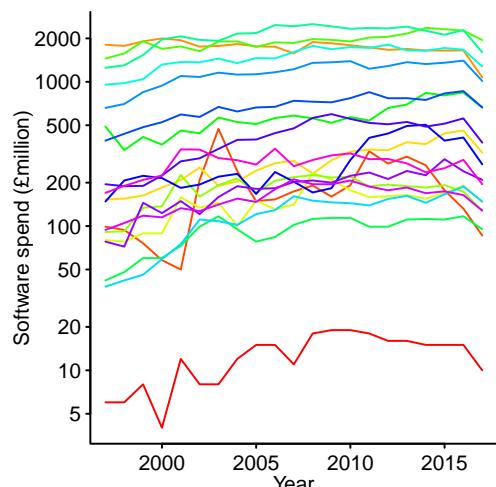


Figure 4.33: Total yearly spend on their own software by the 21 industry sectors in the UK, reported by companies as fixed-assets. Data from UK Office for National Statistics.<sup>1180</sup> [code](#)

Customer influence over the trade-offs associated with software development ranges from total (at least in theory, when the software is being funded by a single customer<sup>viii</sup>), to almost none (when the finished product is to be sold in volume to many customers). One trade-off is the reliability of software vs. its development cost (see Chapter 6); if the customer is operating in a rapidly changing environment, being first to market may have top priority.

Customer demands vary between countries<sup>965</sup> ...

Companies sometimes work together to create a product or service related ecosystem that services their needs. Competing to best address customer need results in product evolution; new requirements are created, existing requirements are modified or may become unnecessary.

A group of automotive businesses created AUTOSAR (Automotive Open System Architecture), with the aim of standardizing communication between the ECUs (Electronic Control Unit) supplied by different companies. A study by Motta<sup>1108</sup> investigated the evolution of roughly 21,000 AUTOSAR requirements, over seven releases of the specification between 2009 and 2015. Figure 4.34 shows the cumulative addition, modification and deletion of the software requirements during this period.

## 4.4.2 Culture

Cultures occur at multiple organizational levels. Countries and regions have cultures, communities and families have cultures, companies and departments have cultures, application domains and products have cultures, software development and programming languages have cultures.

Organizational routines are sometimes encapsulated as grammars of action, e.g., the procedures followed by a support group to solve customer issues.<sup>1233</sup>

The culture (e.g., learned behaviours, knowledge, beliefs and skills) embodied in each developer will influence their cognitive output. Shared culture provides benefits, such as, existing practices (e.g., the set of assumptions and expectations about how things are done) are understood and followed, using categorization to make generalizations from relatively small data sets (see Section 2.5.4). Social learning is discussed in Section 3.4.2.

A shared culture provides an aid for understanding others, it does not eliminate variability. When asked to name an object or action, people have been found to give a wide range of different names. A study by Furnas, Landauer, Gomez, and Dumais<sup>532, 533</sup> described operations to subjects who were not domain experts (e.g., hypothetical text editing commands, categories in *Swap 'n Sale* classified ads, keywords for recipes) and asked them to suggest a name for each operation. The results showed that the name selected by one subject was, on average, different from the name selected by 80% to 90% of the other subjects (one experiment included subjects who were domain experts, and the results for those subjects were consistent with this performance). The frequency of occurrence of the names chosen tended to follow an inverse power law.

Object naming has been found to be influenced by recent experience,<sup>1459</sup> practical skills (e.g., typists selecting pairs of letters that they type using different fingers<sup>1586</sup>) and egotism (e.g., a preference for letters in one's own name or birthday related numbers<sup>856, 1173</sup>).

Separating behavior that is the result of learning rather than developers finding a common solution is difficult. For instance, a study of the use of single letter identifiers in five languages<sup>140</sup> found that *i* was by far the most common in source code written in four of the languages. Is this usage primarily driven by developers abbreviating the words *integer* (the most common variable type) or *index*, or by seeing this usage in example code in books and on the web?...

Cultural differences are a source of confusion and misunderstanding, in dealings with other people, or with their cognitive output...

Metaphors<sup>906</sup> are a figure of speech, which apply the features associated with one concept to another concept. For instance, concepts involving time are often expressed by native English speakers using a spatial metaphor. These metaphors take one of two forms—one in which time is stationary, and we move through it (e.g., “we’re approaching the end of the year”);

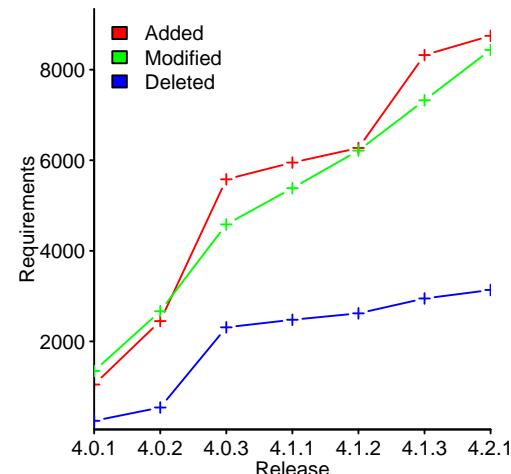


Figure 4.34: Cumulative number of software requirements added, modified and deleted, over successive releases, to the 11,000+ requirements present in release 4.0.0. Data kindly provided by Motta.<sup>1108</sup> [code](#)

<sup>viii</sup> Single customers may be large organizations paying millions to solve a major problem, or a single developer who enjoys creating software.

in the other case, we are stationary and time moves toward us (e.g., “the time for action has arrived”).

A study by Boroditsky<sup>183</sup> investigated subjects’ selection of either the ego-moving, or the time-moving, frame of reference. Subjects first answered a questionnaire dealing with symmetrical objects moving to the left or to the right; the questions were intended to prime either an ego-moving or object-moving perspective. Subjects then read an ambiguous temporal sentence (e.g., “Next Wednesday’s meeting has been moved forward two days”) and were asked give the new meeting day. The results found that 71% subjects responded in a prime-consistent manner; of the subjects primed with the ego-moving frame, 73% thought the meeting was on Friday and 27% thought it was on Monday. Subjects primed with the object-moving frame showed the reverse bias (i.e., 31% and 69%).

Native Chinese speakers also use spatial metaphors to express time related concepts, but use the vertical axis rather than the horizontal axis used by native English speakers.

Cultural conventions can be domain specific; for instance, in the US politicians *run* for office, while in Spain and France they *walk*, and in Britain they *stand* for office. These metaphors may crop up as supposedly meaningful variable names, e.g., `run_for`, `is_standing`.

Is there a software culture shared across many organizational ecosystems, or does the culture of every major development/customer ecosystem evolve independently? Have the ecosystems inhabited by software changed so rapidly that cultural behaviors have not had time to spread and become widely adopted, before others take their place. Distinct development practices did evolve at the three early computer development sites in the UK.<sup>245</sup>

Cultures that operate in proximity are likely to influence each other. Software cultures associated with particular application domains will, over time, adopt aspects of the application domain culture, e.g., character set encodings, which have become embedded in some software cultures. Character encodings have been evolving since the 1870s,<sup>499</sup> with the introduction of the telegraph; early computer capacity restrictions drove further evolution,<sup>1001</sup> and the erosion of these restrictions has allowed a single encoding for all the World’s characters<sup>1579</sup> to slowly spread.

Is English the lingua-franca of software development?

There has been a world-wide diffusion of software systems such as Unix, MS-DOS, and Microsoft Windows, along with the pre-internet era publically available source code, such as X11, gcc, and Unix variants such as BSD; books and manuals have been written to teach software developers about this software. English is the language associated with these software systems and books, which has created incentives for developers to attain good enough English proficiency. A short interval between updates containing major changes of functionality reduces the likelihood that an investment in adapting products to non-English speaking markets may not be worthwhile.

The world-wide spread of popular American culture is another vector for the use of English.

The office suite LibreOffice was originally written by a German company, and many comments were written in German. A decision was made that all comments should be written in English; Figure 4.35 shows the decline in the number of comments contained in the LibreOffice source code, written in German.

Chinese developers required to speak English at work...<sup>1632</sup> emailed for data...

?

In ALGEC,<sup>717</sup> a language invented in the Soviet Union, keywords can be written in a form that denotes their gender and number. For instance, Boolean can be written: ?????????? (neuter), ?????????? (masculine), ?????????? (feminine) and ?????????? (plural). The keyword for the “go to” token is `to`; something about Russian makes use of the word “go” unnecessary.

A fixation on a particular way of viewing the world is not limited to language conventions, it can also be found in mathematics. For instance, WEIRD people have been drilled to use a particular algorithm for dividing two numbers, and believe that the system of number representation used by the Romans caused them serious difficulties with division. Division of Roman numerals is simple if the appropriate algorithm is used:<sup>293</sup> ...

A study by Miu, Gulley, Laland and Rendel<sup>1097</sup> ...

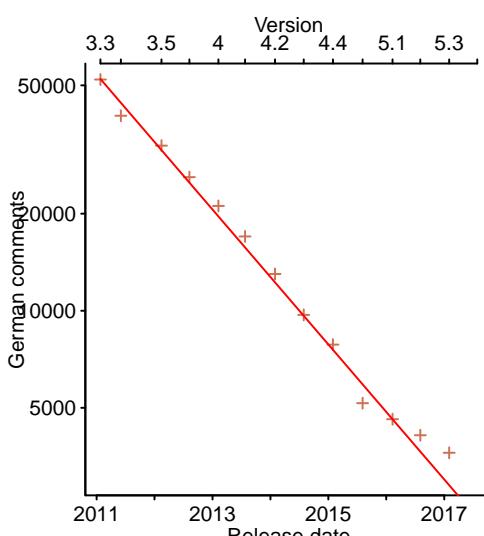


Figure 4.35: Estimated number of comments written in German, in the LibreOffice source code. Data from Meeks.<sup>1059</sup> code

### 4.4.3 Software vendors

A company is a legal entity that limits the liability of the owners. The duty of the directors of a company is to maximise the return on investment to shareholders. Commercial pressure to deliver results in the short term rather than taking a longer term view ...<sup>648</sup>

The first software company, selling software consulting services, was founded in March 1955.<sup>888</sup> Commercial software products, from third-parties, had to wait until computer usage became sufficiently widespread that a profitable market for them was considered likely to exist. One of the first third-party software products ran on the RCA 501, in 1964, and created program flowcharts.<sup>778</sup>

The birth, growth, and death of companies<sup>866</sup> is of economic interest to governments seeking to promote the economic well-being of their country. Company size, across countries and industries, roughly follows a Pareto distribution,<sup>374</sup> while company age has an exponential distribution;<sup>306</sup> most companies are small, and die young. and for software companies?...

Figure 4.36 shows the number of UK companies registered each month having the word software or computer in their SIC description (age distribution coming...).

Top European software companies, by turnover Truffle100/...

Commercial software ecosystems contain companies providing services such as consulting, training and recruitment.<sup>307</sup> Companies who create software systems might be divided into those that create bespoke systems, paid for by a single customer, and those that create a product to sell to multiple customers...

The development of a software system can involve a whole ecosystem of supply companies. For instance, the winner of a contract to develop a large military system often subcontracts-out work for many of the subsystems to different suppliers; each supplier using different computing platforms, languages and development tools (Baily et al<sup>92</sup> describes one such collection of subsystems).

Companies may operate in specific customer ecosystems or within a software ecosystem of a large company might be defined by the platform on which they run, e.g., software running on Microsoft Windows, by the kind of work users of the software do, e.g., making presentations, or by the industry in which users of the software operate, e.g., the travel business...

A study by Crooymans, Pradhan and Jansen<sup>341</sup> investigated the relationship connections between companies in a local software business network; Figure 4.37 shows the relationship links between companies, with a few large companies and many smaller ones.<sup>ix</sup>

### 4.4.4 Career paths

Computers have enabled many new industries to be created, but the stability needed to establish widely recognised software industry specific career paths has not yet occurred. A common engineering career path is various levels of engineering seniority, involving increasing management responsibilities, potentially followed by various levels of purely management activities; the Peter principle may play a role in career progression.<sup>143</sup>

Coding was originally classified as a clerical activity (a significant under-appreciation of the cognitive abilities required), and the gender-based division of working roles prevalent during the invention of electronic computing assigned women to coding jobs; mens' role in the creation of software included non-coding activities such as specifying the formulas to be used.<sup>963</sup>

What are the skills and knowledge that employers seek in employees involved in software development, and what are the personal characteristics of people involved in such occupations?

The U.S., the Occupational Information Network (O\*NET)<sup>1188</sup> maintains a database of information on almost 1,000 occupations, based on data collected from people currently doing the job.<sup>x</sup> The freely available data is aimed at people such as job seekers and HR staff, and includes information on the skills and knowledge needed to do the job, along with personal characteristics and experience of people doing the job.

<sup>ix</sup> Teams of students decided which companies to interview, and so some clustering is the result of students using convenience sampling —email conversation with authors.

<sup>x</sup> At the time of writing the O\*NET website returns 20 matches for software developer occupations; the U.S. Census Bureau maintains its own occupational classification.

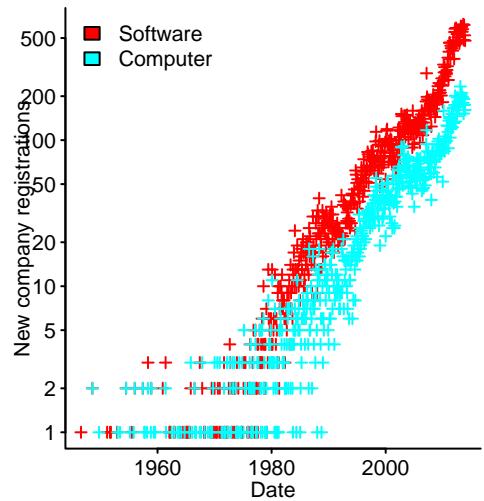


Figure 4.36: Number of new UK companies registered each month, whose SIC description includes the word software or computer (case not significant). Data extracted from OpenCorporates.<sup>1191</sup> code

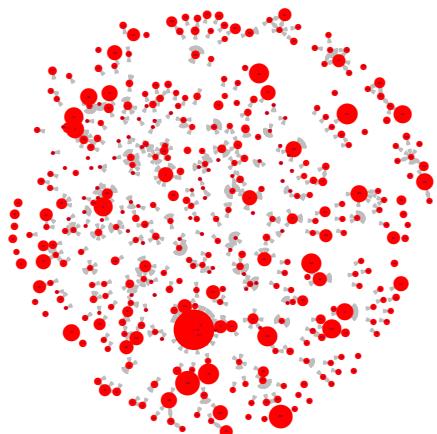


Figure 4.37: Connections between companies in a Dutch software business network. Data kindly provided by Crooymans.<sup>341</sup> code

Software companies employ people to perform a variety of jobs, including management<sup>930</sup>, sales, marketing, engineering, Q/A, customer support, and internal support staff (e.g., secretarial). A study by Jamtveit, Jettestuen and Mathiesen<sup>769</sup> investigated academic research units and found that the ratio of support staff to academic staff was fitted by a power law having an exponent of around 1.30 (a hierarchical management model of one administrator per three sub-units produces an exponent of 1.26). Figure 3.4 shows that even in a software intensive organization around 87% of revenue is spent on non-software development activities.

Employment opportunities are the stepping stones of a career path, and personal preferences will result in opportunities being considered more attractive than others. The popularity of particular software development activities<sup>251</sup> will have an impact on the caliber of potential employees available for selection by employers.

What roles might people having a career in software development fill, what are the common role transitions, how many people are involved, and how long do people inhabit any role?

Census information, and government employment statistics, are sources of data covering people who might be considered to be software developers; however, this data may include jobs that are not associated with software development. A study by Gilchrist and Weber<sup>571</sup> investigated the number of employed computer personnel in the US in 1970. The data for what was then known as *automatic data processing* included keypunching and computer operations personnel; approximately 30% of the 810,330<sup>xi</sup> people appear to have a claim to be software developers (see [ecosystems/50790641-II.R](#)). This data does excludes software development classified under R&D.

Figure 4.38 shows the number of people, stratified by age, employed in the 12 computer related U.S. Census Bureau occupation codes during 2014 (largest peak is the total).

People change, the companies they work for change, and software ecosystems evolve, which creates many of opportunities for people to change jobs,<sup>361</sup> the culture of information technology is one of high staff turnover, compared to other industries.<sup>?</sup>

When companies experience difficulties recruiting people with the required skills, the salaries for the corresponding jobs are likely to increase. An antitrust action was successfully prosecuted<sup>478</sup> against Adobe, Apple, Google, Intel, Intuit, and Pixar for mutually agreeing not to cold-call each others' employees (with a view to them hiring them).

A growing number of companies ask employees to sign non-compete and no-poach agreements.<sup>1492</sup>

A study<sup>174</sup> of manufacturing industry found that family-friendly workplaces were associated with a higher-skilled workforce and female managers; based on the available data, there was no correlation with firm productivity.

Hedonism driven software development ecosystems exist in various forms... Pleasure activism...

Research software engineers<sup>827</sup>...

There are opportunities for developers to make money outside of formal employment...

Figure 4.39 shows a sorted list of the total amount earned by individuals through bug bounties programs. Both studies downloaded data that was available on the HackerOne website; the study by Zhao, Grossklags and Liu<sup>1704</sup> used data from November 2013 to August 2015, and the study by Maillart, Zhao, Grossklags and Chuang<sup>1008</sup> used data from March 2014 to February 2016.

## 4.5 Applications and Platforms

A successful software system attracts its own ecosystem of users and third-party add-ons. The accumulation of an ecosystem may be welcomed and encouraged by the successful owner, or it may be tolerated as a consequence of being successful.

What has been called the Bill Gates line, <sup>1552</sup> provides one method of distinguishing the two cases: "A platform is when the economic value of everybody that uses it, exceeds the value of the company that creates it."

<sup>xi</sup> 127,491 working for the Federal government, 27,839 in state government extrapolated from data on 36 states and estimated 655,000 in private establishments.

Operating in a market where third-parties are tolerated by the apex vendor is a precarious business. Small companies may be able to eek out a living by filling small niches that are not worth the time and effort for the apex vendor, or by having the agility to take advantage of short-term opportunities.

In a few product ecosystems the first release is everything, there is little ongoing market. Figure 4.40 shows the daily minutes spent using an App, installed from Apple's AppStore, against days since first used. This used to be the case when people paid for games software on their phones; a shift to in-game purchases created an ongoing relationship... emailed for data...

In a rapidly moving industry, an ecosystem can effectively provide small companies with resources that exceed those available to much larger companies seeking to do everything themselves...

The rise of open source has made it viable for substantial language ecosystems to flower, or rather substantial package ecosystems, with each based around a particular language. For practical purposes, language choice is now about the quality and quantity of their ecosystem...

### 4.5.1 Platforms

Platform businesses bring together producers and consumers, e.g., shopping malls link consumers and merchants, and newspapers connect readers (assumed to be potential customers) and advertisers; it is a *two-sided* market. Microsoft Windows is the poster child of a software platform.

Platforms create value by facilitating interactions between third-party producers and consumers; which differs from the value-chain model, which creates value by controlling a sequence of activities.

Platform owners aims to maximize the total value extracted of their ecosystem. In some cases this means subsidizing one kind of member to attract another kind (from which a greater value can be extracted). For instance, Microsoft once made development tools (e.g., compilers) freely available to attract developers, who wrote the applications that attracted end-users (there are significantly fewer developers than end-users, and any profit from selling compilers correspondingly smaller).

Value-chains focus on customer value, and seek to maximize the lifetime value of individual customers, for products and services.

Organizations gain an advantage by controlling valuable assets that are difficult imitate. In the platform business the community, and the resources its members own and contribute is a valuable, hard to copy, asset. In a value-chain model these assets might be raw materials or intellectual property.

Building a platform requires convincing third-parties that it is worth joining, ecosystem governance is an important skill.

An organization that can create and own a de facto industry standard does not need to coordinate investments in creating the many enhancements and add-ons; an ecosystem of fragmented third-parties can work independently, they simply need to adhere to an established interface (what provides the coordination); the owner of a platform benefits from the competition between third-parties, who are striving to attract customers by creating desirable add-ons (which enhance the ecosystem, and fills the niches that are not worth the time and effort of the apex vendor).

Platform owners want customers to have a favorable perception of the third-party applications available on the platform. Removing undesirable members<sup>1629</sup>...

Operating systems were the first software ecosystem, and OS vendors offered unique functionality to third-party developers in the hope that they would make extensive use of it in the code they wrote.

Vendors wanting to sell products on multiple operating systems have to decide whether to offer the same functionality across all versions of their product (i.e., not using functionality unique to one operating system to support functionality only available on that OS), or to provide some functionality that varies between operating systems.

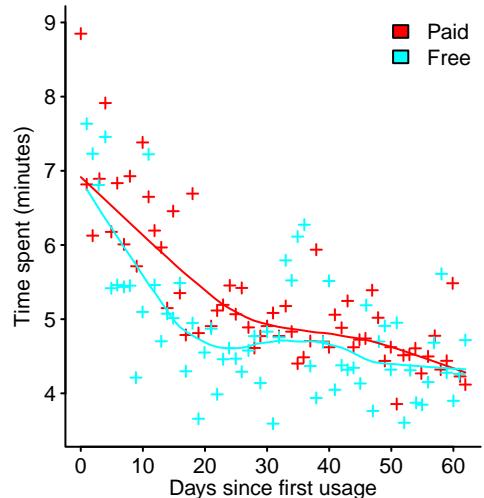


Figure 4.40: Daily minutes spent using an App, from Apple's AppStore (data from 2009). Data extracted from Ansar.<sup>48</sup> code

The term *middleware* is applied to software designed to make it easier to port applications across different operating systems; the Java language, and its associated virtual machine, is perhaps the most well-known example of middleware.

Operating system vendors dislike middleware because it reduces switching costs. Microsoft, with successive versions of Microsoft Windows, was a dominant OS vendor during the 1990s and 2000s, and had an ecosystem control mechanism known as *embrace and extend*. Microsoft licensed Java, and added Windows specific functionality to its implementation, which then failed to pass the Java conformance test suite. Sun Microsystems (who owned Java at the time) took Microsoft to court and won;<sup>1658</sup> Microsoft then refused to ship a non-extended version of Java as part of Windows, Sun filed an antitrust case and won.<sup>1109</sup>

Zynga started as a games producer on Facebook, but then sought to migrate players onto its own platform.

#### 4.5.2 Pounding the treadmill

Once a product ecosystem is established, investment by the apex vendor is focused on maintaining their position, and growing the ecosystem. A recurring trend is for software ecosystems to lose their business relevance, with the apex vendor remaining unchanged; see Section 1.1.

Once up and running, some bespoke software systems become crucial assets for operating a business, and so companies have no choice but to pay whatever it takes to keep them running. From the vendors' perspective, maintenance is the least glamorous, but often the most profitable aspect of software systems; companies sometimes underbid on to win a contract and make their profit on maintenance activities (see Chapter 5...).

Over time, customers' work-flow molds itself around the workings of software products; an organization's established way of doing things evolves to take account of the behavior of the software it uses; staff training is a sunk cost. The cost of changing established practices, real or imaginary, is a moat that reduces the likelihood of customers switching to competing products; it is also a ball-and-chain for the vendor, in that it can limit product updates to those that are not costly for existing customers; at some point the profit potential of new customers may outweigh that of existing customers, and an updated product requires existing customers to make an investment in adapting to the new release.

Pressure to innovate comes from the need to encourage the installed base to upgrade; continual innovation makes it very difficult for potential competitors to create viable alternatives.

Commercial products evolve when vendors believe that investing in updates is economically worthwhile. Updated versions of a product provide justification for asking customers to pay maintenance or upgrade fees, and in a competitive market work to maintain, or improve, market position; product updates also signal to potential customers that the vendor has not abandoned the product (unfavourable publicity about failings in an existing product that could deter potential new customers).

Product version numbers are used to signal information, such as which upgrades are available under a licensing agreement (e.g., updates with changes to the minor version number are included), and as a form of marketing to potential customers (who might view high numbers as a sign of maturity). The release schedule and version of some systems is sufficiently stable that a reasonably accurate regression model can be fitted<sup>1596</sup> (see [regression/release\\_info/co-coon\\_mod.R](#)).

Time between application updates... emailed, pending...?

The regular significant improvement in Intel cpu performance (see Figure 4.14), starting in the last 1980s, became something that was factored into software system development, e.g., future performance improvements were used as a reason for not investing too much effort tuning the performance of the next release...

Files created by users of a product are a source of customer switching costs (e.g., cost of conversion), and vendor ball-and-chain (e.g., the cost of continuing to support files created by earlier versions). File written using a given formats can have very long lifetimes; a common vendor strategy is to continue supporting the ability to read older formats, but only support the writing of more recent formats. A study by Jackson<sup>765</sup> investigated pdf files created using a given version of the pdf specification. Figure 4.42 shows the total number of pdf files created

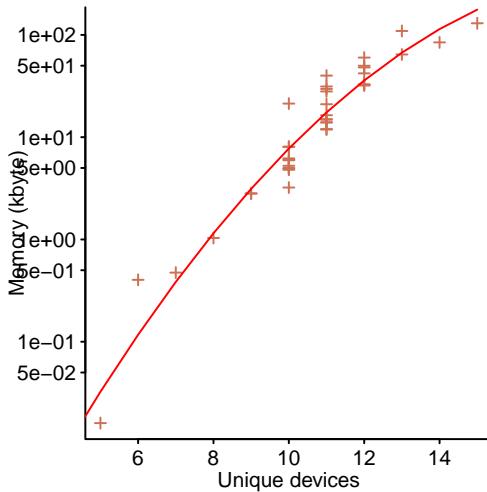
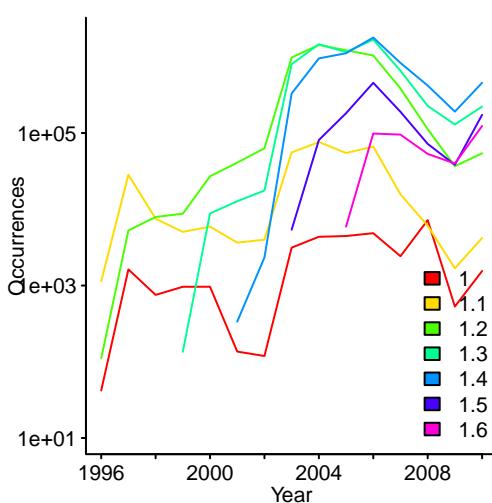


Figure 4.41: Size of 40 operating systems (Kbytes, measured in 1975) capable of controlling a given number of unique devices; line shows a quadratic regression fit. Data from Elci.<sup>440</sup> [code](#)



using a given version of the pdf specification, available on websites having a UK web domain between 1996 and 2010

A few software systems have existed for many decades, and are expected to last many more decades, e.g., simulation of nuclear weapon performance<sup>1269</sup> (the nuclear test-ban treaty prohibits complete device testing).

What are the costs associated with maintaining a software system?

A study by Dunn<sup>420</sup> investigated the development and maintenance costs of 158 software systems developed by IBM (total costs over the first five years); some of these contained a significant percentage of COTS components. The systems varied in size from 34 to 44,070 man-hours of development effort, and involved from 21 to 78,121 man-hours of maintenance. Figure 4.43 shows the ratio of development to average annual maintenance cost. The data is for systems at a single point in time, i.e., 5-years. Modeling, using expected system lifetime, finds that the mean total maintenance to development cost ratio is less than one (see [ecosystems/maint-dev-ratio.R](#)). The correlation between development and maintenance man-hours is 0.5 (0.38-0.63 is the 95% confidence interval, see [economics/maint-dev-cost-cor.R](#)).

Hedonism funded software systems continue to change for as long as those involved continue to enjoy the experience...

?

The issues around fixing reported faults during maintenance are discussed in Chapter 6. In safety critical applications the impact of changes during maintenance has to be thought through;<sup>367</sup> this issue is discussed in Chapter 6.

### 4.5.3 Users' computers

A software system has to be usable within the constraints of the users' hardware, and the installed software libraries.

The days when customers bought their first computer to run an application are long gone. Except for specialist applications and general hardware updates, it is not usually cost effective for customers to invest in new computing hardware specifically to run an application. The characteristics of existing customer hardware is crucial, because software that cannot be executed with a reasonable performance in the customer environment will not succeed. Figure 8.27 shows the variation in basic hardware capacity of desktop systems.

OS, libraries, add-on packages... data... Dependencies of other software systems that have been installed...

The distribution of process execution times often has the form of either power-law or an exponential.<sup>480</sup> In large computer installations, *workload analysis*,<sup>480</sup> analyzing computer usage and balancing competing requirements to maximise throughput, may employ several people. Figure 4.44 shows the distribution of the execution time of 184,612 processes running on a 1995 era Unix computer.

## 4.6 Software development

Most computer hardware is brought because it is needed to run application software,<sup>xii</sup> i.e., software development is a niche activity.

Computers were originally delivered to customers as bare-metal (many early computers were designed and built by established electronics companies<sup>542</sup>); every customer wrote their own software, sometimes obtaining code from other users.<sup>xiii</sup> As experience and customers accumulated,<sup>83</sup> vendors learned about the kinds of functionality that was useful across their customer base.<sup>228</sup> Supplying basic software functionality needed by customers, decreased computer ownership costs, and increased the number of potential customers.

Figure 4.45 shows how the amount of code shipped with IBM computers increased over time. Applications need to receive input from the outside world, and produce output in a

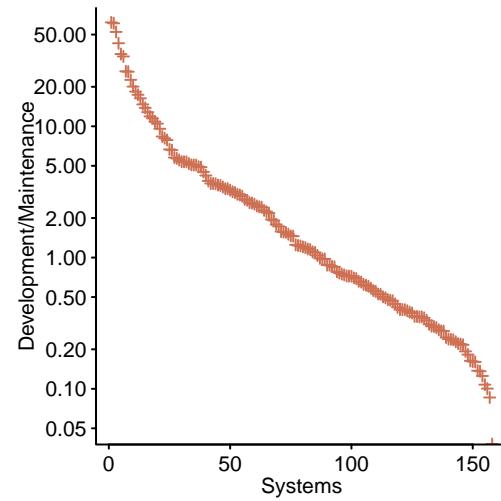


Figure 4.43: Ratio of development costs to average annual maintenance costs (over 5-years) for 158 IBM software systems sorted by size; curve is a beta distribution fitted to the data (in red). Data from Dunn.<sup>420</sup> [code](#)

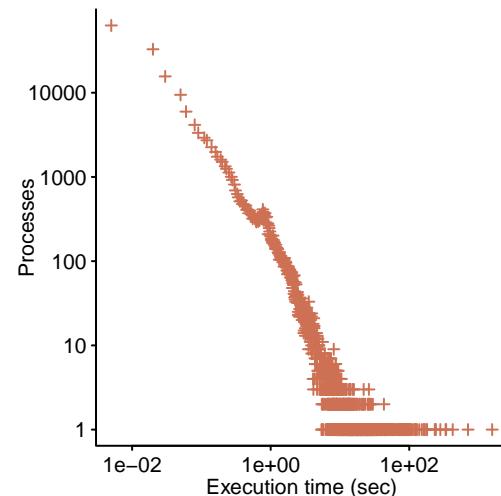


Figure 4.44: Number of Unix processes executing for a given number of seconds, on a 1995 era computer. Data from Harchol-Balter et al.<sup>656</sup> [code](#)

<sup>xii</sup> There have been cases of companies buying hardware without realising that applications were not included.<sup>54</sup>

<sup>xiii</sup> Software user-groups are almost as old as computers.<sup>54</sup>

form acceptable to their users; interfacing to many different peripherals is a major driver of OS growth, see Figure 4.41 (also [?]).

All software for the early computers was written using machine code, which made it specific to one particular kind of computer. New computers were constantly being introduced, each having different characteristics (in particular machine instructions).<sup>169, 1056, 1649, 1650</sup>

Machine independent programming languages (e.g., Fortran in 1954<sup>82</sup> and Cobol in 1961<sup>138</sup>) were created with the aim of reducing costs. The cost reduction came through reducing the dependency of program source code on the particular characteristics of the hardware on which it executed, and reuse of programming skills learned on one kind of computer to different computers (e.g., removing the need to learn the assembly language for each new machine).

## 4.6.1 Programming languages

Organizations with an inventory of source code have an interest in the current and future usage of programming languages: more widely used languages are likely to have more developers capable of using it (reducing hiring effort), likely to have more extensive tool support (than less widely used languages), and are likely to be available on a wider range of platforms.

Humans have an innate desire to create language,<sup>?</sup> writers of science fiction sometimes go to great length to create elaborate, and linguistically viable, languages for aliens to speak.<sup>1348</sup> Factors found to influence the number of human languages actively used, within a geographic region, include population size, country size, and the length of the growing season,<sup>1146</sup> see [ecosystems/009.R](#).

The specification of Plankalkül, the first high-level programming language, was published in 1949,<sup>116, 1718</sup> the specification of a still widely used language (Fortran) was published in 1954.<sup>745</sup> A list of compilers<sup>xiv</sup> for around 25 languages was published in a book<sup>608</sup> published in 1959 (see [ecosystems/Grabbe\\_59.txt](#)), and in 1963 it was noted<sup>665</sup> that creating a programming language had become a fashionable activity. A 1976 report<sup>500</sup> estimated that at least 450 general-purpose languages and dialects were currently in use within the US DoD.

Many thousands of programming languages have been created, only a handful have become widely used. Figure 4.46 shows the number of new programming languages, per year, that have been described in a published paper.

High-level languages did not immediately displace machine code for the implementation of many software systems. A 1977 survey<sup>1484</sup> of programmers in the US Federal Government, found 45% with extensive work experience, and 35% with moderate work experience, of machine code. Developing software using early compilers could be time-consuming and labor-intensive; memory limits required compiling to be split into a sequence of passes over various representations of the source (often half-a-dozen or more<sup>211</sup>), with the intermediate representations being output on paper-tape, which was read back in as the input to the next pass (after reading the next compiler pass from the next paper-tape in the sequence), eventually generating assembler. Some compilers required that the computer have an attached drum-unit<sup>84</sup> (an early form of hard-disk), which was used to store the intermediate forms (and increase sale of peripherals). Developer belief in their own ability to produce more efficient code than a compiler was also a factor.<sup>83</sup>

The creation of a major new customer facing ecosystem provides an opportunity for new languages and tools to become widely used within the development community working in that ecosystem. For instance, a new language might provide functionality often needed by applications targeting ecosystem users, that is more convenient to use (compared to preexisting languages), alternatively there may only be one language initially available for use, e.g., Solidity for writing Ethereum contracts.

Within geographical, or linguistic, separated regions the popularity of existing languages has sometimes evolved along different paths, e.g., Pascal remained popular in Russia<sup>1010</sup> longer than elsewhere, and in Japan Cobol remains widely used<sup>20</sup>...

The term *language popularity* suggests that the users of the language have some influence on the selection process, and like the language in some way. In practice developers may not have any say in the choice of language, and may have no positive (or negative) feelings towards

<sup>xiv</sup> The term compiler was not always applied in the sense that is used today.

the language. However, this term is in common use and there is nothing to be gained by using something different...

Figure 4.47 shows the number of lines contained in the source code of the 32 major releases of Debian, stratified by programming language (the legend lists the top five languages).

Sources of information on language usage include:

- Job adverts: The number of times organizations appear to be willing to pay money to someone to use a language, is both a measure of actual usage and perceived popularity. Languages listed in job adverts are chosen for a variety of reasons including: appearing trendy in order to attract young developers to an interview, generating a smokescreen to confuse competitors, and knowledge of the language is required for the job advertised.

Social media includes postings to employment websites and adverts for jobs. Figure 4.48 shows the number of monthly developer job related tweets that included a language name. Recruitment agencies... emailed for data... Computer Weekly data...

- quantity of existing code: the total quantity of existing code might be used as a proxy for the number of developers who have worked with a language in the past (see Figure 11.10); recent changes in the quantity of existing code is likely to provide a more up-to-date picture.

Quantity of code on github...

- number of books sold:<sup>679</sup> spending money on a book is an expression of intent to learn the language. The intent may be a proxy for existing code (i.e., learning the language to get a job, or work on a particular project), existing course curriculum decisions, or because the language has become fashionable.

Sales data from individual publishers is likely to be affected by the popularity of their published books, and those of competing publishers; see [ecosystems/LangBooksSold.R](#),

- miscellaneous: prior to the growth of open source, the number of commercially available compilers was an indicator of size of the professional developer market for the language.

Question & answer websites... language complexity, questions tail off as answers on common issues are become available. Figure 4.49 shows the normalised percentage of language related tags associated with questions appearing on Stack Overflow each month.

Application domain specific usage, such as mobile phone development,<sup>1312</sup> and embedded systems in general<sup>1576</sup>...

US Federal government surveys of its own usage: a 1981 survey<sup>320</sup> found most programs were written in Cobol, Fortran, Assembler, Algol and PL/1, a 1995 survey<sup>719</sup> of 148 million LOC in DOD weapon systems Ada represented 33%, the largest percentage of any language (C usage was 22%)...

Books are a source of programming languages related discussion going back many decades. Language names such as Fortran and Cobol are unlikely to be used in non-programming contexts, while names such as Java and Python are more likely to be used in a non-programming context. Single letter names, such as C, or names followed by non-alphabetic characters have a variety of possible interpretations, e.g., the phrase *in C* appears in music books as a key signature, also the OCR process sometimes inserts spaces that were probably not in the original. The English unigrams and bigrams from the Google books project<sup>1075</sup> are available...

A study by Savić, Ivanović, Budimac and Radovanović<sup>1377</sup> investigated the impact of a change of teaching language on student performance in practical sessions (going from Modula-2 to Java). Student performance, measured using marks assigned, was unchanging across the four practical sessions, as was mean score for each year; see [ecosystems/2016-sclit-uup.R](#) for details.

Some programming languages evolve, i.e., new features are added and existing features are removed. One consequence of language evolution is that some implementations may not support constructs contained in the source code used to build a program<sup>145</sup>...

Applications can only be ported to a platform when compilers for the languages used are available. The availability of operating systems and device drivers written in C, means that a C compiler is one of the first developer tools created for a new processor.

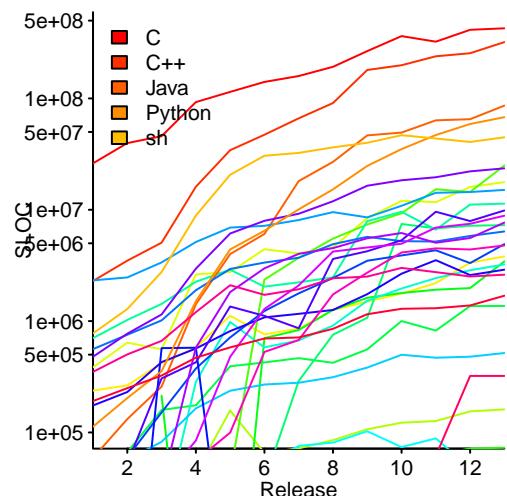


Figure 4.47: Lines of code written in the 32 programming languages appearing in the source code of the 13 major Debian releases between 1998 and 2019. Data from the Debsources developers.<sup>377</sup> [code](#)

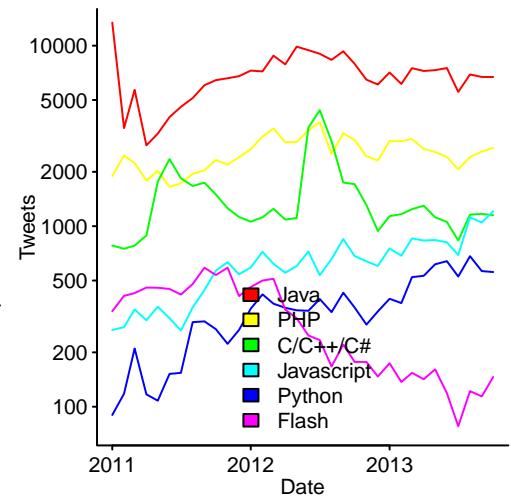


Figure 4.48: Number of monthly developer job related tweets specifying a given language. Data kindly provided by Destefanis.<sup>395</sup> [code](#)

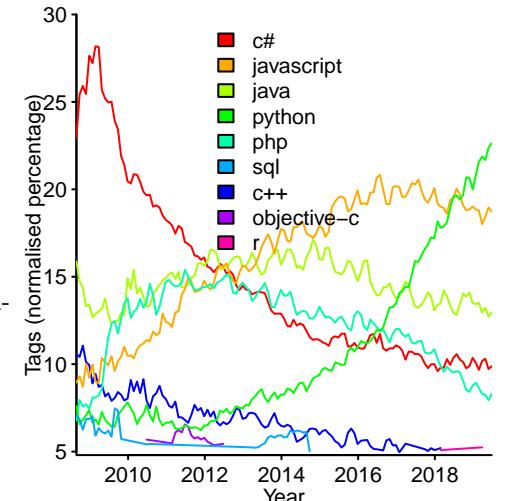


Figure 4.49: Normalised percentage of 34 language tags associated with questions appearing on Stack Overflow in each month. Data extracted from Stack Overflow.<sup>7</sup> [code](#)

Will today's widely used languages continue to be reasonably as popular over the next decade (say)?

Implementing a software system is expensive; it is often cheaper to continue working with what exists. The quantity of existing source contained in commercially supported applications is likely to be a good indicator of continuing demand for developers capable of using the implementation language(s). existing use of the language, with the survival rate of applications written in it

Perception (i.e., not reality) within developer ecosystems of which languages are considered worthwhile knowing, becoming popular or declining in usage/popularity.<sup>1072</sup> In a rapidly changing environment developers want to remain attractive to employers, there is a desire to have a CV that lists experience in employable languages,

Reid<sup>1321</sup> investigated the first language used in teaching computer science majors; between 1992 and 1999 over 20 language lists were published, with an average of 400+ universities per list. More recent surveys<sup>1437</sup> have been sporadic. Figure 4.50 shows the percentage of languages used by at least 3% of the responding universities. The main language taught in universities during the 1990s (i.e., Pascal) ceased being widely used in industry during the 1980s.

Compiler vendors enhance their products by supporting features not specified in the language standard (if one exists). The purpose of adding these features is to attract customers (by responding to demand), and over time to make it more difficult for customers to switch to a different vendor. Some language implementations become influential because of the widespread use of the source code they compile (or interpret). The language extensions supported by an influential implementation have to be implemented by any implementation looking to be competitive for non-trivial use.

A study by Rigger, Marr, Adams and Mössenböck<sup>1332</sup> investigated the use of compiler specific built-in functions supported by gcc, in 1,842 Github projects. In C and C++ source code, built-ins have the form of functions, but are subject to special processing by the compiler. Analysis of the C compiler source found 12,339 distinct built-ins (some were internal implementations of other built-ins), and 4,142 unique built-ins were encountered in the C projects analysed. Figure 4.51 shows the cumulative growth in the number of Github projects supported, as support is added for more built-ins (the growth algorithm assumed an implementation order based on the frequency of unique built-in usage across projects)...

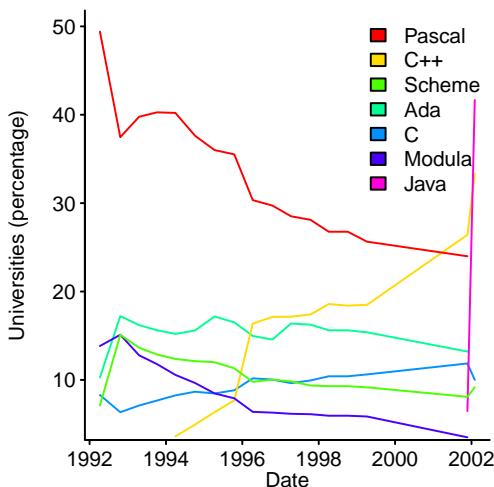


Figure 4.50: Percentage of universities reporting the first language used to teach computer science majors. Data from Reid, via the Wayback Machine.<sup>1321</sup> [code](#)

## 4.6.2 Libraries and packages

Program implementation often makes use of functionality that is found in many other programs, e.g., opening a file and reading from it, and writing to a display device. Developers working on the first computers had access to a library of commonly used functions,<sup>1667</sup> and over time a variety of motivations have driven the creation and supply of libraries.

Some programming languages were designed with specific uses in mind, and included support for application domain library functions, e.g., Fortran targeted engineering/scientific users and required support for trigonometric functions, Cobol targeted business users, and required support for sorting.<sup>xv</sup>

Manufacturers discovered that the libraries they had bundled with the operating system<sup>168</sup> to attract new customers, could also be used to increase customer lock-in (by tempting developers with extensive library functionality having characteristics specific to the manufacturer, that could not be easily mapped to the libraries supplied by other vendors).

The decreasing cost of hardware, and the availability of an operating system, in the form of Unix source code, enabled many manufacturers to enter the minicomputer/workstation market. Vendors' attempts to differentiate themselves led to the Unix wars<sup>1372, 1373</sup> of the 1980s (in the mid-1990s, platforms running a Unix-derived OS typically shipped with over a thousand C/C++ header files<sup>785</sup>).

The POSIX standard<sup>759</sup> was intended provide the core functionality needed to write portable programs; this functionality was derived from existing implementation practice of widely used Unix systems.<sup>1717</sup> POSIX became widely supported (in part because large organizations, such as the US Government, required vendors to supply products that included POSIX

<sup>xv</sup> The C Standard specifies support for some surprising functions, surprising until it is realised that they are needed to implement a C compiler, e.g., `strtoul`.

support, although some implementations felt as-if they were only intended to tick a box during a tender process, rather than be used).

A study by Atlidakis, Andrus, Geambasu, Mitropoulos and Nieh<sup>65</sup> investigated POSIX usage (which defines 1,177 functions) across Android 4.3 (1.1 million apps measured, 790 functions tracked, out of 821 implemented) and Ubuntu 12.04 (71,199 packages measured, 1,085 functions tracked, out of 1,115 implemented). Figure 4.52 illustrates how the relative usage of POSIX functions can vary across application domains (these numbers do not include calls to `ioctl`, whose action is to effectively perform an implementation defined call).

Linux came late to the Unix wars, and emerged primary OS. The Linux Standard Base<sup>xvi</sup> is intended to support a binary compatibility interface for application executables; this interface includes pseudo-file systems (e.g., `/proc`) that provide various kinds of system information. A study by Tsai, Jain, Abdul and Porter<sup>1565</sup> investigated use of the Linux API by Ubuntu 15.04 applications, including system calls and pseudo-file system usage... schema promised...

The internet created a new method of software distribution: a website. People and organizations have built websites to support the distribution of packages available for a particular language (rather than a particular OS), and some have become the primary source of third-party packages for their target language, e.g., npm for Javascript, and CRAN for R.

Publicly available package repositories are a relatively new phenomena; a few were started in the mid-1990s (e.g., CRAN and CPAN), and major new repositories were still appearing 15-years later (e.g., Github in 2008 and npm in 2010). The age of a repository play a large role in an analysis of the characteristics measured, e.g., rate of change is likely to be high in the years immediately after a repository is created.

Package repositories are subject to strong network effects. Developers want to minimise the effort invested in searching for packages, and the repository containing the most packages is likely to attract the most searches; also, the number of active users of a repository is a signal for authors of new packages, seeking to attract developers, who need to choose a distribution channel.

A study by Caneill and Zacchiroli<sup>249</sup> investigated package usage in the official Debian releases and updates. Figure 4.53 shows the survival curve of the latest version of a package included in 10 official Debian releases, along with the survival of the same version of a package over successive releases.

A niche market may be large enough, and have sufficiently specialist needs, for a repository catering to its requirements to become popular within the niche, e.g., the Bioconductor website aims to support the statistical analysis of biological assays and contains a unique collection of R packages targeting this market.

Figure 4.54 shows the number of R packages in three large repositories (a fourth, R-forge is not displayed), along with the occurrences of packages in multiple repositories (colored areas not scaled by number of packages). Each website benefits from its own distinct network effects, e.g., Github provides repositories, and gives users direct control of their files.

The requirements, imposed by the owners of a repository, for a package to be included, may add friction to the workflow of a package under active development (e.g., package authors may want the latest release to be rapidly available to potential users). For instance, some R packages under active development are available on GitHub; with updates being submitted to CRAN once they are stable (some package have dependencies on packages in the other repositories, and dependency conflicts exist<sup>380</sup>).

npm and possible commercial use of repo usage of data...

In some cases a strong symbiotic relationship between language and package ecosystem has formed, which has influenced the uptake of new language revisions, e.g., one reason for the slowed uptake of Python 3 has been existing codes' use of packages that require the use of Python 2...

The installation of a package may fail because of dependency conflicts between the packages already installed and this new package, e.g., installed package  $P_1$  may depend on package  $P_2$  having a version less than 2.0, while package  $P_3$  depends on package  $P_2$  having a version of at least 2.0. A study by Drobisz, Mens and Di Cosmo<sup>415</sup> investigated Debian package

<sup>xvi</sup> LSB 3.1 was first published as an ISO Standard in 2006, it was updated to reflect later versions of the LSB in... The Linux API has evolved.<sup>89</sup>

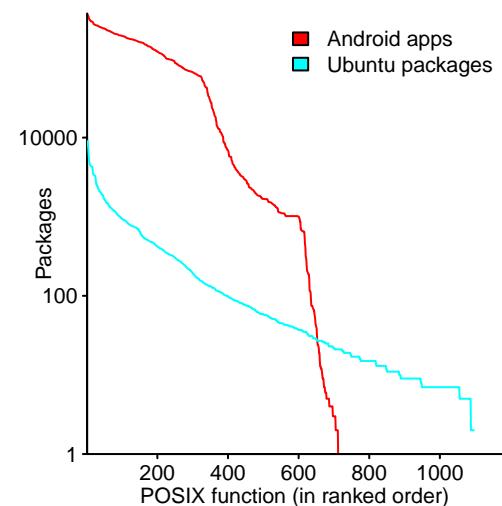


Figure 4.52: Number of Android/Ubuntu (1.1 million apps)/(71,199 packages) linking to a given POSIX function (sorted into rank order). Data from Atlidakis et al.<sup>65</sup> code

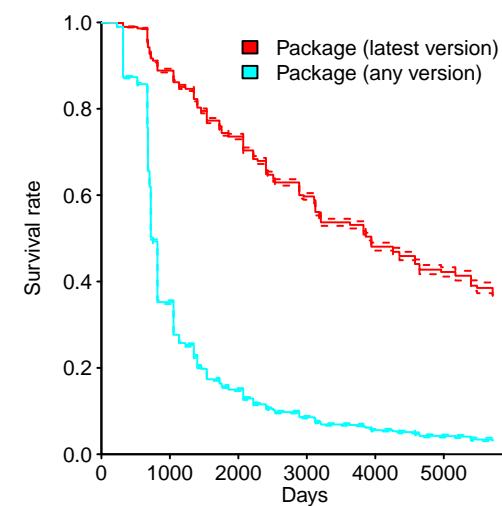


Figure 4.53: Survival curve of packages included in 10 official Debian releases, and inclusion of the same release of a package. Data from Caneill et al.<sup>249</sup> code

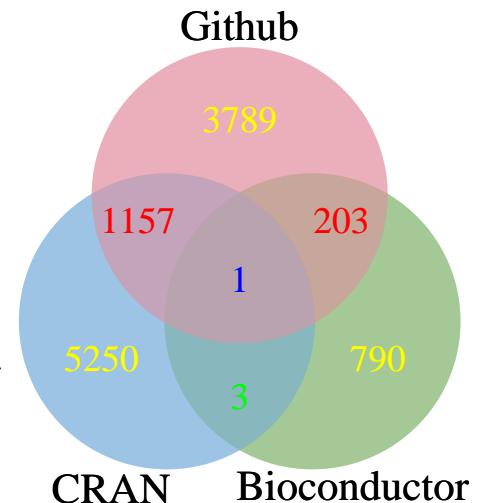


Figure 4.54: Number of packages in three widely used R repositories (during 2015), overlapping regions show packages appearing in multiple repositories (areas not to scale). Data from Decan et al.<sup>379</sup> code

dependency conflicts. Figure 4.55 shows the survival curve of package lifetime, and the interval to a package's first conflict.

Package dependency conflicts only become an issue if an attempt is made to install the packages with conflicting dependencies... A study by Decan, Mens and Claes<sup>378</sup> investigated how package dependencies in three ecosystems (npm, CRAN and RubyGems), changed over time.... [ecosystems/SANER-2017.R](#)

Use of semantic versioning to provide developers with information about backwards compatibility...?

emailed for info on columns...?

Third-party libraries may contain undocumented functionality that is accessible to developers. This functionality may be intended for internal library use, and is only accessible because it is not possible to prevent developers accessing it; or, the vendor intent is to provide privileged access to internal functionality for their own applications, e.g., Microsoft Word making use of undocumented functionality in the libraries supplied with Microsoft Windows.

There may be a short term benefit for developers making use of internal functionality, that makes it worth the risk of paying an unknown cost later if the internal functionality is removed or modified; see Figure ???. Information on undocumented functionality in Microsoft Windows was widely available,<sup>1394</sup> with major Windows updates occurring every three years, or so.

Major updates to Android have occurred once or twice a year (see Figure 8.36), and the lifetime of internal functionality has been declining exponentially (see Figure 4.56).

### 4.6.3 Tools

Computer manufacturers were the primary suppliers of software development tools until suppliers of third-party tools had cost-effective access to the necessary hardware, and the potential customer base became large enough to make supplying tools commercially attractive.

When developer tools are sold for profit, vendors have an incentive to keep both customer and the tool users happy.<sup>xvii</sup> Open source has significantly reduced the number of customers for some tools, while maintaining similar numbers of users for these tools. For instance, a company developing their own cpu might fund the implementation of a code generator, based on an open source compiler, for this processor, with the resulting compiler being made freely available; the users of this new compiler are the actual product.

The workflow used for software development influences the desired characteristics of the tools used. For instance, during the early years of computing, interaction with computers was usually via batch processing, rather than one-line access;<sup>583</sup> users submitted a job for the computer to perform (i.e., a sequence of commands to execute), and it joined the queue of jobs waiting to be run. Developers might only complete one or two edit/compile/execute cycles per day. In this environment, high quality compiler error recovery can significantly boost developer productivity; having an executable for an error corrected version of the submitted source (along with a list of errors found), gives developers a binary that may be patched to do something useful.<sup>xviii</sup> Borland's Turbo-Pascal stopped compiling at the first error it encountered, dropping the user into an editor with the cursor located at the point the error was detected. Compilation was so fast, within its interactive environment on a personal computer, developers loved using it.

Adding support for new language features, new processors and environments is a source of income for compiler vendors; the performance of code generated by compilers are often benchmarked, to find which generates the faster and/or more compact code. Compile time options provide a means for users to tailor compiler behavior to their needs.

Figure 4.57 shows the number of options supported by gcc for specific C and C++ options, and various compiler phases, for 96 releases between 1999 and 2019; during this time the total number of options supported grew from 632 to 2,477.

Support for an option is sometimes removed; of the 1,091 non-processor specific options supported, 214 were removed before release 9.1. Since 1999, the official release of GCC has included support for 80 cpus, of which 20 were removed before release 9.1 (see Figure 4.12).

<sup>xvii</sup> Customers pay money, users use; a developer can be both a customer and a user.

<sup>xviii</sup> Your author once worked on a compiler project, funded with the aim of generating code 60% smaller than the current compiler. Developers hated this new compiler because it generated very little redundant code; the redundant code generated by the previous compiler was useful because it could be used to hold patches.

#### 4.6.4 Information sources

Readily available sources of reliable information can significantly reduce the costs of learning.

Vendors that support an API have to document the calls available and provide examples, if they want developers, third-party or otherwise, to make use of the services provided. APIs evolve, and the documentation has to be kept up to date.<sup>1429</sup> The growth of popular APIs, over time, can result in an enormous collection of documentation, e.g., the 130+ documents for the Microsoft Server protocol specifications<sup>1076</sup> contain over 16 thousand pages (see Figure 8.24).

Which parts of an API document are read most often?...? fig 5 emailed...

Q/A sites... accuracy of answers, correct answers that are not correct for later product releases...

Development mailing lists...

Books and articles... marketing material written by individuals offering consultancy or training services. A high quality book, or manual, may reduce the size of the pool of potential clients, but those remaining are made aware of a potential high quality supplier of consultancy.

Coworkers as sources for established ways of doing things, libraries, tools (e.g., build systems and source code control, available third-party libraries), question/answer sites, forges, and a collection vendors making a living from supporting developer activities (e.g., offering training, third-party libraries and compilers)...

Standards produced by various organizations...

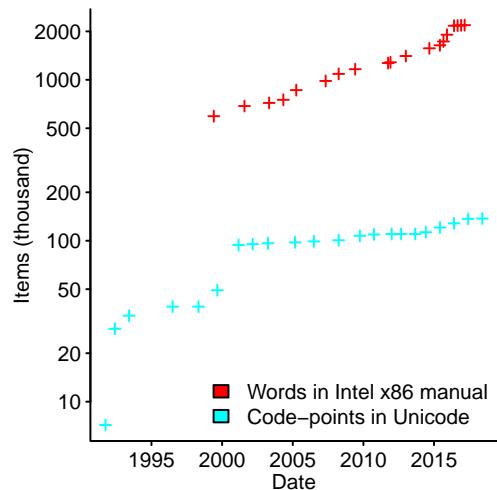


Figure 4.58: Words in Intel x86 architecture manuals, and code-points in Unicode Standard over time. Data for Intel x86 manual kindly provided by Baumann.<sup>117</sup> [code](#)



# Chapter 5

## Projects

### 5.1 Introduction

The hardest part of any project is, generally, obtaining the funding to implement it.<sup>1396</sup>

Clients<sup>i</sup> are paying for a solution to a problem and have decided that a software system is likely to provide a cost effective solution. The client might be a large organization contracting a third-party to develop a new system, a company writing software for internal use, or an individual spending their own time implementing an idea they have.

Some projects are started by an individual, working to satisfy a personal need, who then attracts other developers<sup>683</sup>...

Successfully implementing a software system involves creating a financially viable implementation that solves the client's problem. Financial viability means not so expensive the client is unable to pay for it and not so cheap that the vendor fails to make a reasonable profit.

A commercial software project aims to implement the clients' understanding of the world (in which they operate, or want to operate), in a computer executable model that can be integrated into the business and, economically operated.

Useful programs vary in size from a few lines of code to millions of lines and might be written by an individual in under an hour or by thousands of developers over many years. Much of the existing research has focused on large projects, reasons for this include: the bureaucracy needed to support large projects creates paperwork from which data can be extracted for research, and the organizations providing the large sums needed to finance large projects are able to influence research agendas. This book is driven by the availability of data and much of this data comes from large projects; small projects are important and because they are much more common than large projects, it is possible they are economically more important.

What are the characteristics of the majority of software projects? Figure 5.1 suggests that most are completed in under a year, contain less than 40 KSLOC and that much of the effort is performed by external contractors (data from a multi-year data collection process<sup>20</sup> by the Software Engineering Center, of Japan's Information-Technology Promotion Agency).

"The first go-around at it was about \$750 million, so you figure that's not a bad cost overrun for an IT project. Then I said, "Well, now, tell me. Did you do an NPV? Did you do a ROI? What did you do on a \$750 million IT investment?" And she sort of looked a little chagrined and she said, "Well, actually, there was no analysis done on that." I said, "Excuse me . . . can you explain that to me please. That's not what the textbook says." She said, "Well, it was a sales organization, the brokers worked for the sales organization." The sales organization —this was a few years ago when the brokerage business was extremely good—said, "you know, the last two years we've made more than enough money to pay for this. We want it, and we're going to pay for it." And the board of directors looked at how much money they were making and they said, "You go pay for it". So that was the investment analysis for a \$750 million IT investment that turned into a billion dollars."<sup>1532</sup>

It is unwise to ask clients why they want the software. Be thankful that somebody is willing to pay to have bespoke software written, creating employment for software developers.

<sup>i</sup> The term *customer* has mass market associations, bankrolling bespoke software development deserves something having upmarket connotations.

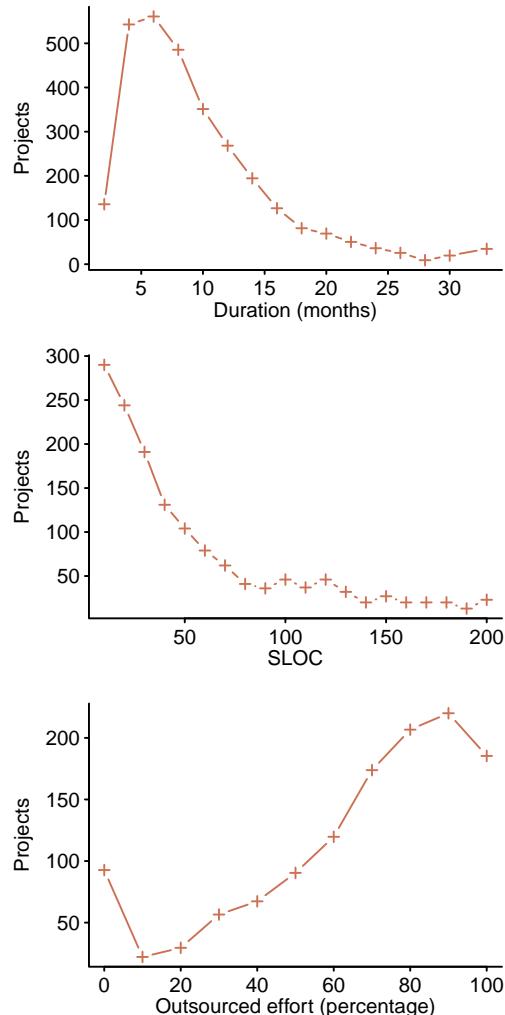


Figure 5.1: Number of projects having a given duration (upper; 2,992 projects), delivered containing a given number of SLOC (middle; 1,859 projects), and using a given percentage of out-sourced effort (lower; 1,267 projects). Data extracted from Akita et al.<sup>20</sup> code

Shrink-wrapped software products have a recommended retail price; finding out how much the client is willing to pay is an important part of the bespoke software sales process. James Webb, then head of NASA, told President Kennedy that \$20 billion was the price tag for landing on the Moon (NASA engineers had previously given Webb an estimated cost of \$10–12 billion,<sup>1403</sup> the final cost was \$24 billion).

The first important question a vendor needs to answer, when asked by a potential client about creating a bespoke software system, is: does the client have the money needed to pay for such a project? If the client appears to have the money and is willing to spend it, the vendor may consider it worth investing, to get a good-enough view on whether what the client thinks they want can be implemented well enough for them to pay for an implementation.

A study by Anda, Sjøberg and Mockus<sup>40</sup> sent a request to tender to 81 software consultancy companies in Norway, 46 did not respond with bids. Figure 5.2 shows the estimated schedule days and bid price received from 14 companies (21 companies provided a bid price without a schedule estimate). Fitting a regression model (making use of information on an assessment of the analysis and design performed by each company, along with estimated planned effort) explains almost 90% of the variation in the schedule estimates; see [projects/effort-bidprice.R](#).

Projects live for as long as they are funded and can only exceed their original budget when the client is willing to pay. A project that misses its delivery date only continues to live because the client continues to agree to pay for delivery.

For some projects, clients have no choice but to pay what it takes and wait for delivery,<sup>1054</sup> since not having a working software system is likely to result in the client ceasing to be competitive, resulting in ruin or a loss significantly greater than the cost of paying for the software. The funding for some projects is driven by company politics<sup>148</sup> or senior management ego, and a project might be cancelled for reasons associated with how it came to be funding, independently of any cost/schedule issues.

Skyscrapers are like software projects in that each one is a one-off project, sharing many implementation details with other skyscrapers but also having its own unique implementation features. Figure 5.3 shows that the rate of construction of skyscrapers (in meters per year), has remaining roughly unchanged.

Both the client and the vendor want their project to be a success. What makes a project a success?

From the vendor perspective (i.e., this book’s point of view) a successful project is likely to be one that produces an acceptable profit<sup>ii</sup>. A project may be a success from the vendor’s perspective and be regarded as a failure by the client (because the client paid, but did not use the completed system,<sup>17</sup> or because users under-utilised the delivered system or avoided using it<sup>928</sup>) or by the developers who worked on the project (because they created a workable system despite schedules and costing underestimates by management<sup>970</sup>). These different points of view mean that the results from project success surveys<sup>36</sup> are open to multiple interpretations.

A study by Milis<sup>1080</sup> investigated views of project success by professionals in the roles of management, team member (either no subsequent project involvement after handover, or likely to receive project benefits after returning to their department), and end-user. Fitting regression models to the data from 25 projects, for each role, finds one explanatory variable of project success common to all roles: user happiness with the system. Management considered being within budget as an indicator of success, while other roles were more interested in meeting the specification; see [projects/Milis-PhD.R](#) for details.

A study by Garman<sup>546</sup> surveyed 70 program and project managers for US Air Force projects about their priorities. Meeting expectations (according to technical specifications) was consistently prioritized higher than being on budget or on time; none of the available explanatory variables could be used to predict priority decisions; see [projects/ADA415138.R](#) for details.

A project may fail because the users of a system resist its introduction into their workflow<sup>919</sup> (e.g., they perceive its use as a threat to their authority). Failure to deliver a system can result in the vendor having to refund any money paid by the client and make a payment for work performed by the client.<sup>66</sup>

<sup>ii</sup> Research papers often use keeping within budget and schedule, as measures of success; this begs the question of which budget, and which schedule, e.g., the initial, final, or intermediate versions?

A study by Zwikaal and Globerson<sup>1719</sup> of project cost and schedule overruns, and success in various industries, including software, suggests that except for the construction industry, overrun rates are broadly similar.

Software development is a punctuated information arrival process. Particular development activities may have dominant modes of information discovery.

Those involved in software projects, like all other human activities, sometimes engage in subversion and lying.<sup>1353</sup> Activities range from departmental infighting to individual rivalry, and motivations include: egotism, defending one's position, revenge, or a disgruntled employee.

Almost as soon as computers became available million line programs were being written to order. Figure 5.4 shows lines of code and development costs for US Air Force software projects, by year, excluding classified projects and embedded systems; the spiky nature of the data suggests that LOC and development costs are counted in the year a project is delivered.

### 5.1.1 Project pecking-order

Software projects are implemented within an existing cultural framework, which assigns some level of power and authority to those involved. A project's level of power and authority decides the extent to which outsiders have to adapt to the needs of the project, or the project has to be adapted to the ecosystem in which it is being created. The following are some common development culture frameworks:

- one-off projects within a company, which treats software as a necessary investment, which has to be made to keep some aspect of the business functioning. Company employees involved in the development might treat working on the project as a temporary secondment that is part of what they perceive to be their main job, perhaps a stepping stone to a promotion, or a chance to get hands-on experience developing software (even with a view to doing this full time). External contractors may be hired to work on the project for some fixed period,
- projects within a company, which derives most of its income from software products, e.g., software developed to be sold as a product. In the case of startups the founding project initiates the culture, Conway's law of organization...
- projects within a company, where software is the enabler of the products and services, which produce the income, e.g., embedded systems.

A study by Powell<sup>1274</sup> investigated software projects for engine control systems at Rolls-Royce. Figure 5.5 shows effort distribution (in person hours) over four projects, plus non-project work (blue) and holidays (purple'ish, at the top), over 20 months. Staff turnover and use of overtime during critical periods, means that total effort changes over time (also see Figure 11.69).

In its 2015 financial year Mozilla, a non-profit company that develops and supports the Firefox browser, had income of \$421,275 million and spent \$214,187 million on software development.<sup>1516</sup> Almost all of this income came from Google, who paid to be the default search engine used by Firefox,

- a series of one-off projects paid for by another company for internal use, i.e., contract software development. Companies in the contract software development business need to keep their employees busy with fee earning work and may assign them to distribute their time across multiple projects,
- projects where the income is the enjoyment derived from working on it; the implementation is part of the developers' lifestyle.

Single person projects are likely to experience greater variation than larger projects, not because the creator is driven by hedonism, but because a fixed release criteria may not exist and other activities may cause work to be interrupted for indefinite periods of time... Data driven by one source, rather than coming from many, is likely to show greater variance...

A few single developer projects grow to include hundreds of paid and volunteer developers. The development work for projects such as Linux<sup>328</sup> and GCC is spread over multiple organizations, making it difficult to estimate the level of commercial funding.

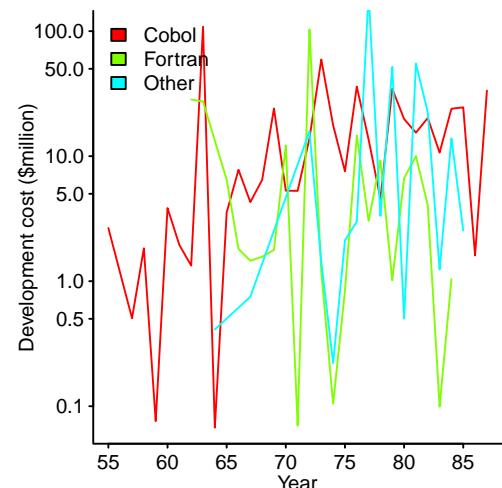
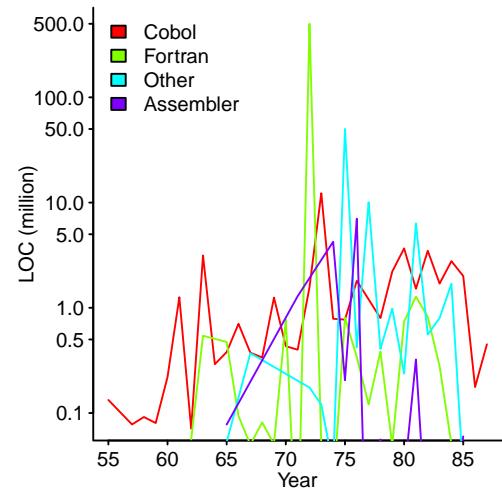


Figure 5.4: Yearly development cost and lines of code delivered to the US Air Force between 1960 and 1986. Data extracted from NeSmith.<sup>1145</sup> [code](#)

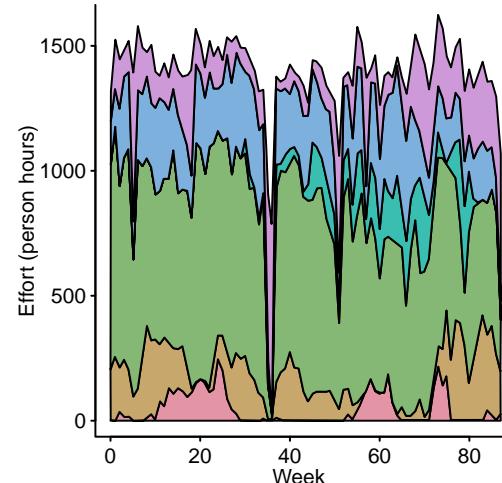
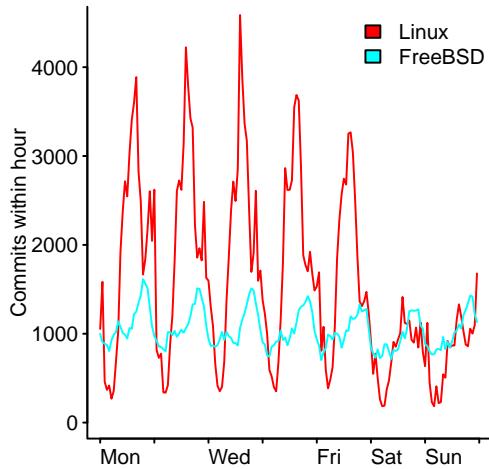


Figure 5.5: Distribution of effort (person hours) during the development of four engine control systems projects, plus non-project work and holidays, at Rolls-Royce. Data extracted from Powell.<sup>1274</sup> [code](#)

Code commits made by volunteer developers are less likely to occur during the working week than commits made by paid developers. Figure 5.6 shows the hourly commits during a week (summed over all commits) for Linux and FreeBSD, suggesting that Linux has many more paid developers.



The software project may be producing one component of many, with it being necessary for those on the project to work with other projects to build a larger integrated system, e.g., a spacecraft.

Academic software projects might fall within any of these categories (with personal reputation being the income sought<sup>1118</sup>).

### 5.1.2 Cancellation

A non-trivial percentage of projects, software and non-software,<sup>1053</sup> are cancelled without ever being used. Open source and personal projects are not cancelled as-such, people simply stop working on them.<sup>308</sup> The cost-effectiveness of any investment decision (e.g., investing to create software that is cheaper to maintain) needs to include the possibility that the project may be cancelled.

- a survey by El Emam and Koru<sup>439</sup> of 84 midlevel and senior project managers, between 2005 and 2007, found the majority reporting IT project cancellation rates in the range 11-40%; Whitfield<sup>1657</sup> lists 105 outsourced UK government related ICT (Information and Communication Technology) projects between 1997 and 2007, of which 30% were terminated,
- the 1994 CHAOS report<sup>1486</sup> is a commonly cited survey of project cost overruns and cancellations. This survey is not representative of projects because it explicitly focuses on failures, subjects were asked: “ . . . to share failure stories.”, i.e., the report lists many failures and high failure rates because subjects were asked to provide this very information. The accuracy of the analysis used to calculate the summary statistics listed in the report has been questioned.<sup>461,806</sup>

Lifecycle stage	Cancelled	Completed	Overrun (schedule and/or cost)
Feasibility	0	214	0
Requirements analysis	3	211	0
Design	28	183	32
Code	15	168	57
Testing	4	164	57
Implementation	1	163	69
Handover	0	163	69
Total	23.8%	76.2%	

Table 5.1: Project cancellation rates by development stage. Data from McManus et al.<sup>1049</sup>

## 5.2 Resource estimation

People tend to be overconfident, and analysis shows that there are benefits to being overconfident; overconfidence is an evolutionary stable cognitive bias (see Section 2.8.5).

People cling to the illusion that it's reasonable to make estimates about the future, even though they have not measured anything in the past...

What kind of estimate is needed, how much opportunity is there to change it later, and what are the client expectations (e.g., minimum credible and maximum willing to spend limits)?

When making a purchase decision in everyday life, people have an expectation that the seller can and will provide information on cost and delivery date, it is not considered unreasonable to expect specific answers.

Bespoke software development is not a service that many clients regularly fund; most clients want a cost and delivery date agreed before they hand over any money or sign a contract, because they perceive this to be the way things are done. Occasionally projects appear to have money thrown at them, with no fixed delivery dates attached. In these cases, the cost is likely to be very small compared to the expected benefits, but there is always a day of reckoning.

A client is paying for bespoke software because nothing suitable is available. How much will it cost to build a system that has never been implemented before, by those involved? Perhaps something very similar already exists, but unless vendors are aware of its existence and using it has a better ROI (to them), compare to implementing something new, they have no incentive to suggest the client use it.

The following are some incentives influencing client and vendor resource estimates:

- estimating is a social process (see Figure 2.59), people often want to get along with others, which means prior estimation information can have an impact<sup>51</sup> (see Figure 5.7); client expectations have an anchoring effect on cost estimates,<sup>807</sup> and during a workshop on anchoring attendees exhibited less of an anchoring effect on an estimation task<sup>1428</sup> (see [projects/DeBiasExptData.R](#)),
- pressure is applied to planners<sup>1622</sup> to ensure their analysis presents a good case for project funding. A former president of the American Planning Association said:<sup>513</sup> “I believe planners and consultants in general deliberately underestimate project costs because their political bosses or clients want the projects. Sometimes, to tell the truth is to risk your job or your contracts or the next contract . . .”
- short-termism: get something out there, and if enough people find it useful money for ongoing work will be found. Projects are regularly cancelled, or fail to deliver worthwhile benefits, taking short-cuts to minimise upfront costs can be a cost effective strategy...
- client unwillingness to pay for detailed analysis of proposed projects; there is no incentive for vendors to invest in detailed analysis until a contract is signed.

Believability...?

Inaccurate project resource estimates are endemic in industry<sup>1069</sup> (and perhaps to all human activities), software projects are just one instance. A study by Flyvbjerg, Holm and Buhl<sup>514</sup> of 258 transportation projects (worth \$90 billion) found costs are underestimating in around 90% of projects, and an average cost overrun of 28% (sd 39); a study of 35 major US DOD acquisitions<sup>179</sup> found a 60% average growth in total costs; an analysis of 12 studies,<sup>718</sup> covering over 1,000 projects, found a mean estimation error of 21%; Butts and Linton<sup>238</sup> discuss, in detail, overruns on the development of over 150 NASA spacecraft.

While the reasonableness of wanting a cost estimate cannot be disputed, an analysis of the number of unknowns involved, and the experience of those involved leads to the conclusion that may be unreasonable to expect accurate resource estimates.

There is lots of evidence<sup>iii</sup> that most resource estimates are inaccurate, with the estimate often being much lower than what is actually used. Figure 5.8 shows estimated time against actual for project sizes over four-orders of magnitude.

Cost overruns are not something the buyer or seller are likely to be want to publicise, and there are probably many more instances than publicly reported.

Software project cost overruns may be blamed on poor project management,<sup>933</sup> but the poor management may have occurred during estimation. The Queensland Health Payroll System Commission of inquiry report<sup>285</sup> provides a detailed analysis of a large failed project.

?

A study by Grimstad and Jørgensen<sup>628</sup> investigated the consistency of estimates made by the same person; seven developers were asked to estimate sixty tasks, over a period of three months; unknown to them, everybody estimated (in work hours) six tasks twice. Figure 5.9 shows the first/second estimates for the same task made by the same subject; identical first/second estimates appear on the grey line and estimates for identical tasks have the same color (and extent of some color clustering shows agreement between developers).

<sup>iii</sup> Whitfield<sup>1657</sup> lists 105 outsourced UK public sector ICT (Information and Communication Technology) projects between 1997 and 2007, having a total value of £29.6 billion; 57% of contracts experienced cost overruns (totalling £9.0 billion) with the average cost overrun being 30.5%, and 30% of contracts were terminated.

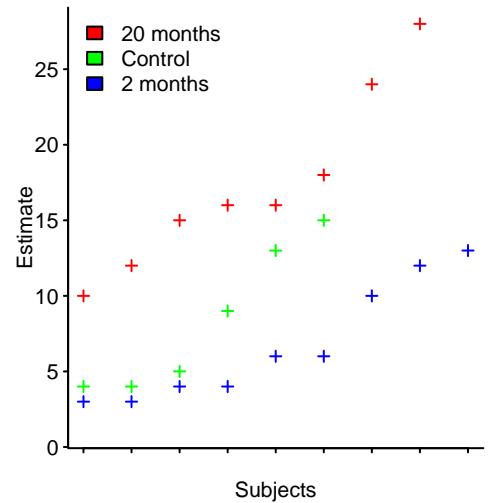


Figure 5.7: Estimate given by three groups of subjects after seeing a statement by a middle manager containing an estimate (2 months or 20 months) or no estimate (control). Data from Aranda,<sup>51</sup> [code](#)

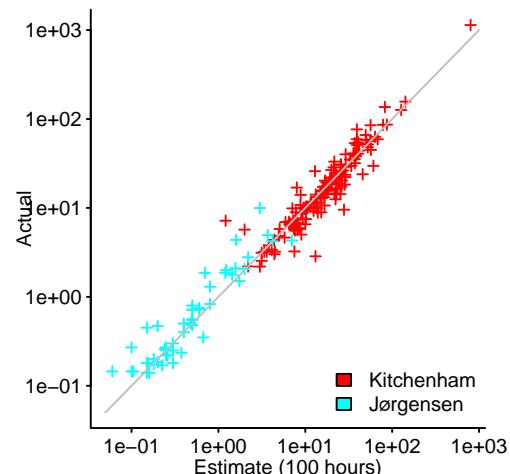


Figure 5.8: Estimated and actual project implementation effort. Data from Jørgensen<sup>797</sup> and Kitchenham et al.,<sup>857</sup> [code](#)

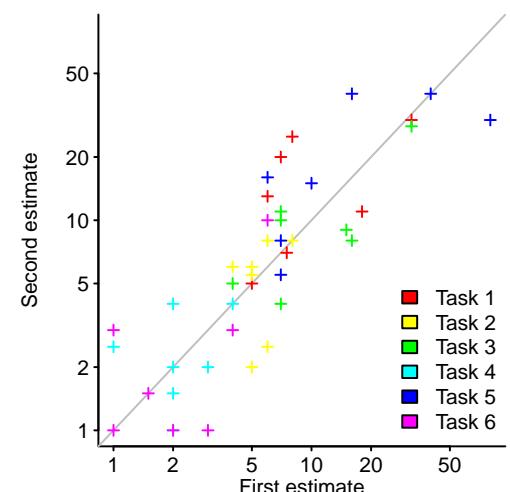


Figure 5.9: Two estimates (in work hours), made by seven subjects, for each of six tasks. Data from Grimstad et al.,<sup>628</sup> [code](#)

A study by Boehm, Gray and Seewaldt<sup>176</sup> investigated the impact of prototyping on the implementation effort of a small application (between 2 and 4 KLOC). Seven teams of students implemented the same specification...

The principles of cost estimation<sup>7</sup> are the same whatever the kind (i.e., software or otherwise) and size of project. The size of the project will have a strong influence on the amount of effort invested in forming an accurate estimate, along with the skill of those doing the estimation. For economically large projects it may be cost-effective to involve one or more expert cost estimators.

The nature of software systems development makes accurate estimation of the implementation resources needed a nonsensical goal. Duplicating everything that was done during a previous successful project implementation will deliver a project within a known cost and schedule, however, duplicating everything will result in duplicate output and there are much cheaper ways of duplicating software.

A new software system is being created because no existing system is available to do the job. The reliability of any cost and time estimates depends on the accuracy with which all the necessary tasks are enumerated, and the accuracy with which the resources needed to implement these tasks can be estimated.

Enumerating necessary tasks entails first enumerating client requirements (discussed below); the risk of changing requirements is a source of resource uncertainty...

Information about the resources previously used to implement the same or similar tasks is essential input to the resource estimating process for pending task; they provide a guide to the lower bounds of the resources that might be needed. When direct information is not available, indirect information is sometimes used e.g., lines of code have been used to provide a mapping to costs (see Figure 3.26).

When little, or no, information is available, following the herd can be an effective strategy.

A study by Wang and Zhang<sup>1633</sup> investigated the distribution of effort, in man-hours, used during the five major phases of 2,570 projects (the types of project were: 20% new development, 68% enhancement, 7% maintenance, 5% other projects). Figure 5.10 shows a density plot of the effort used in each phase, as a fraction of total project effort; the terminology in the legend follows that used by the authors (a mapping of those terms differing from Western usage might be: Produce is implementation, Optimize is testing and Implement is deployment). The distribution of means and medians did not vary with project duration.

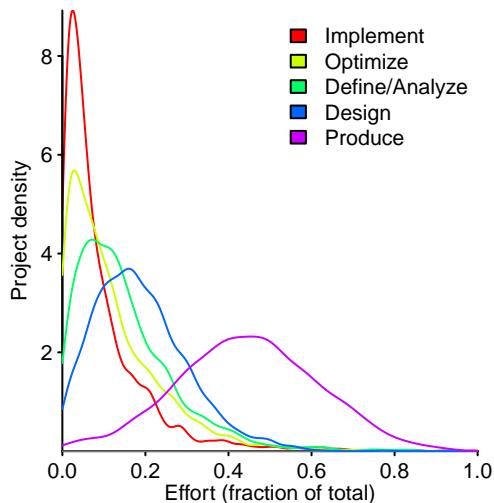


Figure 5.10: Density plot of number of projects investing a given fraction of their total effort in a given project phase. Data kindly provided by Wang.<sup>1633</sup> [code](#)

Figure 5.11 shows the mean and median effort spent on projects taking a given number of months to complete (total of 2,103 projects); the lines are fitted quadratic equations. Assuming 150 man-hours per month, project team size appears to have increased from one to perhaps six or seven, as project duration increased.

Resource estimation is a knowledge based skill acquired through learning from others and practical experience (see the expertise subsection for a discussion of this issue). Making use of the prior project implementation experience of those doing the estimate is a fall-back position, when nothing better is available. It would be surprising if people could estimate the resources needed to implement something that is completely new to them.

Reusing significant amounts of code complicates cost and schedule estimation.<sup>321</sup>

In some situations there is an incentive to over-estimate the time needed to complete a project. For instance, most employees are not in the position of having to undercut competitors to be given work to do. Unless the employee is under pressure, when asked by a manager, how long a task will take it may be better to over-estimate and delivery early, than to under-estimate and delivery late. The equation  $actual = estimate^{0.67}$ , explains almost half the variance in the data for Figure 8.17.

[projects/HuijgensPhD.R...?](#)

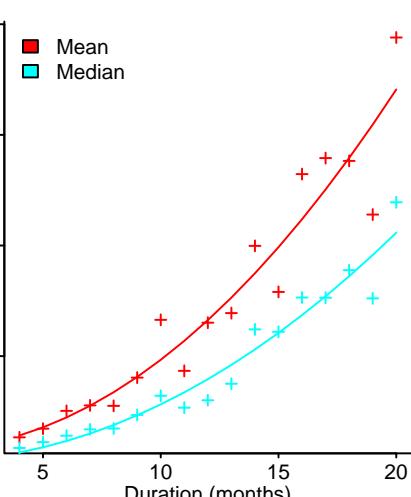


Figure 5.11: Mean and median effort (hours) for projects having elapsed time between four and 20 months (lines are fitted quadratics). Data from Wang et al.<sup>1633</sup> [code](#)

## 5.2.1 Estimation models

Estimation of computer programming costs has proved to be a complex process, with early studies<sup>471, 472</sup> identifying over fifty factors, a 1966 management cost estimation handbook<sup>1143</sup> contained a checklist of 94 questions (based on an analysis of 169 projects). Approaches that

have been used to build software project effort estimation models include: finding equations that best fit data from earlier projects, deriving and solving theoretical models and building project development simulators.

**Fitting data:** Early cost estimation models used this approach.<sup>1651</sup> The problem with fitting equations to data, is that the resulting models are only likely to perform well when estimating projects having the same characteristics as the projects from which the data used to fit the model, was obtained. A study by Mohanty<sup>1101</sup> compared the estimates produced by 12 models; Figure 5.12 shows how widely the estimates varied.

A study by Ourada<sup>1200</sup> evaluated four effort estimation models used for military software (two COCOMO derivatives REVIC and COSTMODL, plus SASET and SEER; all fitted to data); reached the conclusion: “I found the models to be highly inaccurate and very much dependent upon the interpretation of the input parameters.” Similar conclusions were reached by Kemerer<sup>832</sup> who evaluated four popular cost estimation models (SLIM, COCOMO, Function Points and ESTIMACS) on data from 15 large business data processing projects; Ferens<sup>488</sup> compared 10 models against DOD data and came to the same conclusion, as did another study using data from ground based systems.<sup>657</sup>

**Deriving equations:** Norden<sup>1165</sup> studied development projects, which he defined as “ . . . a finite sequence of purposeful, temporarily ordered activities, operating on a homogeneous set of problem elements, to meet a specified set of objectives . . . ”: completing a project involves solving a set of problems ( $W(t)$ ) is the proportion of problems solved at time  $t$ ) and these problems are solved by the people resources ( $p(t)$ , which encodes information on number of people and their skill) is ; the rate of problems solving depends on the number of people available, and the number of problems remaining to be solved:

$$\frac{dW}{dt} = p(t)[1 - W(t)]$$

whose solution is:  $W(t) = 1 - e^{- \int^t p(\tau) d\tau}$

If the skill of the people resource grows linearly, as the project progresses, i.e.,  $p(t) = at$ , the work rate is:

$$\frac{dW}{dt} = ate^{-at^2/2}$$

This equation is known, from physics, as the Rayleigh curve. Putnam<sup>1289</sup> evangelised the use of Norden’s model for large software development projects. Criticism of the Norden/Putnam model has centered around the linear growth assumption (i.e.,  $p(t) = at$ ) being unrealistic.

An analysis by Parr,<sup>1215</sup> modeled the emergence of problems during a project as a binary tree and assumed that enough resources are available to complete the project in the shortest possible time. Under these assumptions the derived work rate is:

$$\frac{dW}{dt} = \frac{1}{4} \operatorname{sech}^2 \frac{\alpha t + c}{2}, \text{ where sech is the hyperbolic secant: } \operatorname{sech}(x) = \frac{2}{e^x + e^{-x}}.$$

While these two equations look very different, the fitted curves they produce are similar; one difference is that the Rayleigh curve starts at zero, while the Parr curve starts at some positive value.

A study by Basili and Beane<sup>108</sup> investigated the quality of fit of various models to six projects requiring around 100 man-months of effort. Figure 5.13 shows Norden-Putnam, Parr and quadratic curves fitted to effort data for project 4.

Both the Norden/Putnam and Parr equations can be derived using hazard analysis,<sup>1603</sup> with the lifetime of problems to be discovered having a linear and logistic hazard rate respectively.

The derivation of these, or any other equation, is based on a set of assumptions, e.g., a model of problem discovery (the Norden/Putnam and Parr models assume that there are no significant changes to the requirements), and the necessary manpower can be added or removed at will. The extent to which the derived equation apply to a project depends on how closely the characteristics of the project meet the assumptions used to build the model.

**Simulation models:** A simulation model that included all the major processes involving in a development project could be used to estimate the resources likely to be needed. There have been several attempts to build such models using Systems Dynamics.<sup>3,227,1004</sup> Building a simulation model requires understanding the behavior of all the important factors involved and as this book shows, we are a long way from having this understanding.

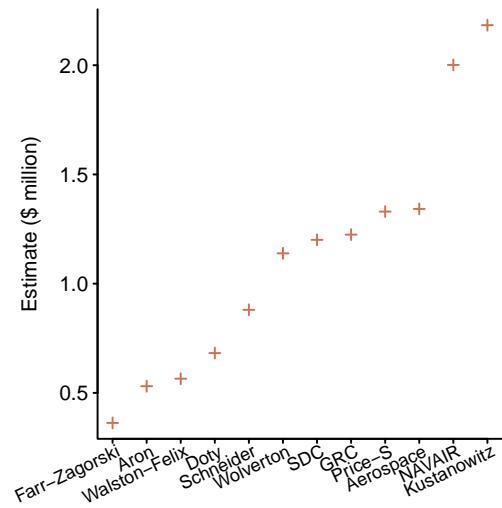


Figure 5.12: Estimated project cost from 12 estimating models. Data from Mohanty.<sup>1101</sup> code

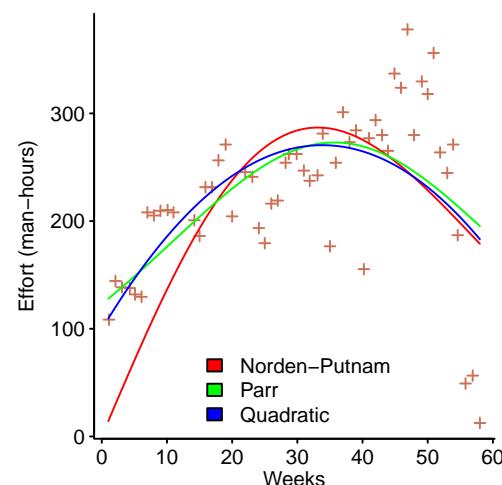


Figure 5.13: Elapsed weeks (x-axis) against effort in man-hours per week (y-axis) for a project, plus three fitted curves. Data extracted from Basili et al.<sup>108</sup> code

**Requirements based** Various techniques have been developed to estimate the resources needed to implement a set of detailed requirements. Examples include: function-points (various counting algorithms have been active used), use cases and story points.

\#NoEstimates, ?

A study by Kampstra and Verhoef<sup>818</sup> investigated the reliability of function point counts. Figure 5.14 shows normalised cost for 149 projects, from one large institution, having an estimated number of function points; also see [projects/82507128-kitchenham](#).

A study by Huijgens and van Solingen<sup>733</sup> investigated two projects considered to be best-in-class, out of 345 projects, from three large organizations. Figure 5.15 shows the cost per requirement, function point and story point for these two projects over 13 releases.

A study by Commeyne, Abran and Djouab<sup>317</sup> investigated effort estimates made using COSMIC function-points and story-points. Figure 5.16 shows the estimated number of hours needed to implement 24 story-points, against the corresponding estimated function-points.

Excessive accuracy can cause experts to question the expertise of those providing the estimate<sup>979</sup> ...

?

## 5.2.2 Money

Companies in the business of bespoke software development have to make a profit on average, over all the projects undertaken, i.e., they have some flexibility in over- and underestimating the cost of individual projects. Figure 5.17 shows the percentage loss/profit made by a software house on 146 fixed-price projects.

A study by Hill, Thomas and Allen<sup>699</sup> investigated the accuracy of over 500 task time estimates made by six managers. Around two-thirds of tasks were over estimated, one third underestimated, and 7% were on target. Over all 500+ tasks, the mean estimate was 13.51 days, actual 13.63, with standard deviations 27.83 and 27.2 respectively.

The total cost of project implementation includes the salaries of those people directly working on it, plus overheads such as office rental, support staff and equipment costs. A study by Jørgensen<sup>798</sup> selected six vendors, from 16 proposals received from a tender request, to each implement the same database-based system. The estimated price per work-hour ranged from \$9.1 to \$28.8 (mean \$13.85).

In an attempt to reduce costs some companies have offshore the development of some projects, i.e., awarded the development contract to a company in another country. A study by Šmite, Britto and van Solingen<sup>1580</sup> of outsourced project costs, found additional costs outside of the quoted hourly rates; these were attributable to working at distance, cost of transferring the work and characteristics of the offshore site. For the projects studied, likely break-even, compared to on-shore development, was thought to be several years later than planned. A study by Deutsch and Jørgensen<sup>396</sup> attempted to calculate the offshoring costs generated by what they labeled *psychic distance* (a combination of differences including cultural, language, political, geographic, and economic development).

## 5.2.3 Time

When will the system be ready for use? This is the client question that comes before or immediately after the cost request.

People time is invested to build a system; people having a minimum set of required skills and a degree of dedication. This issue is discussed in Section 5.5.

Taking longer than estimated usually means greater costs...

In some cases time and money are interchangeable on a project,<sup>949</sup> but there may be dependencies that prevent time (or money being spent) until some item is available. A cost plus contract provides an incentive to spend money, with time only being a consideration if the contract specifies penalty clauses.

Client may need to use the software by a given date...

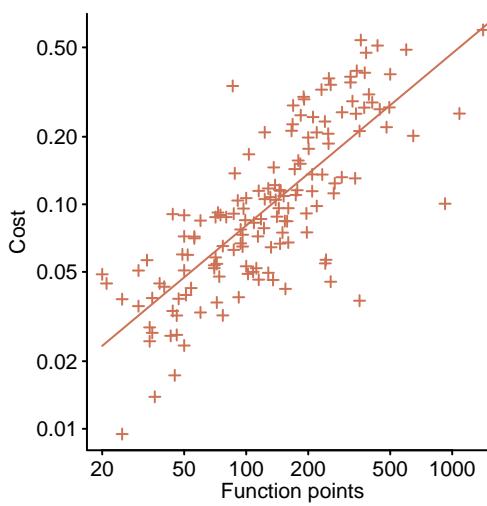


Figure 5.14: Function points and corresponding normalised costs for 149 projects from one large institution. Data extracted from Kampstra et al.<sup>818</sup> [code](#)

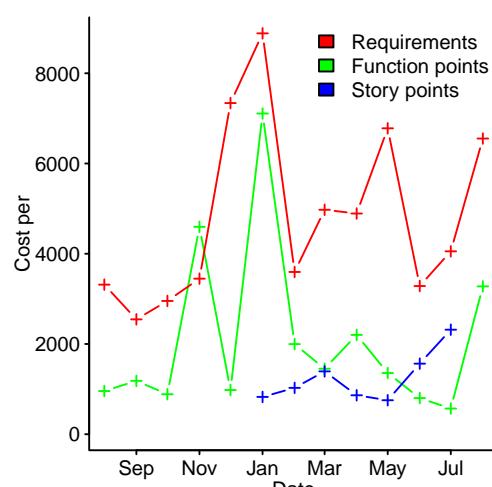
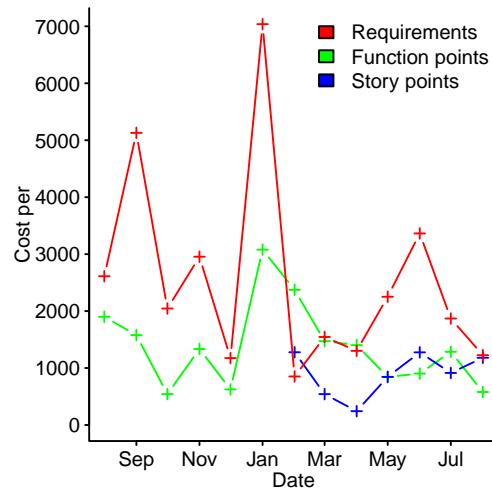


Figure 5.15: Cost per requirement, function point and story point for two projects, over 13 monthly releases. Data from Huijgens.<sup>733</sup> [code](#)

A study by Moløkken-Østvold, Jørgensen, Tanilkan, Gallis, Lien and Hove<sup>1104</sup> investigated 44 software project estimates made by 18 companies. Fitting a regression model to the data finds that estimated project duration (in elapsed days or work hours), along with kind of client (i.e., internal/external), estimation technique used and method of project organization had a statistically significant impact on actual project duration; see [projects/RK31-surveycostestim.R](#).

A study by Kitchenham, Pfleeger, McColl and Eagan<sup>857</sup> investigated the estimation accuracy of 145 projects; none of the factors measured, other than first estimate, were good predictors of actual effort; see [projects/82507128-kitchenham](#). A study by Jørgensen<sup>797</sup> investigated developer estimates for client driven product changes within a medium-sized web-development company, along with a brief description on the reason for difference between estimated/actual effort; see [projects/Regression-models.R](#).

emailed, no response...?

## 5.2.4 Size

Program size is an important consideration when computer memory capacity is measured in kilobytes. During the 1960s mainframe computer memory was often measured in kilobytes, as was minicomputer memory in the 1970s and microcomputer memory in the 1980s. Today, low cost embedded controls might contain a kilobyte of memory, or less, e.g., electronic control units (ECU) used in car engines contain a few kilobytes of flash memory.

Figure 5.18 shows IBM's profit margin on sales of all System 360s sold in 1966 (average monthly rental cost, for 1967, in parentheses). Low-end systems, with minimal memory, were sold below cost to attract customers (and make it difficult for competitors to gain a foothold in the market<sup>390</sup>).

A study by Lind and Heldal<sup>971</sup> investigated the possibility of using COSMIC function points to estimate the compiled size of software components used in ECUs. Function points were counted for existing software components in ECUs used by Saab and General Motors; Figure 5.19 shows COSMIC function points against compiled size of components from four ECU modules; lines are a fitted regression model for each module. While a consistent mapping exists for components within each module, the function point counting technique used did not capture enough information to produce consistent results across modules.

How much variation is to be expected in the size of programs (perhaps measured in lines) implementing the same functionality, in the same language?

Figure 9.13 shows the number of lines contained in over 6,300 C programs implementing the  $3n + 1$  problem (mean lines of code is 46). A more substantial example is the number of statements and declarations (comments or blank lines were not counted) in five Pascal compilers targeting the same mainframe.<sup>1432</sup>

Figure 5.20 shows data from seven studies where multiple implementations of the same specification were implemented in the same language. The fitted regression (grey line) finds, to a good approximation, that standard deviation is one quarter of the mean. With so much variation, in LOC, between different implementations, a wide margin of error must be assumed for any estimation technique based on lines of code.

Multiple teams developing the same project in different languages...?

Lines of code are the end result of decisions made at higher levels of abstraction, where different implementations intended to perform similar activities vary in how their internal components are structured.<sup>867</sup>

## 5.2.5 Software reuse

Reusing existing code has the potential to save time and money, and reused code may contain fewer faults than newly written code (i.e., if the reused code has been executed, there has been an opportunity for faults to be experienced and fixed). Many equations detailing the costs and benefits of software reuse have been published,<sup>1079</sup> what they all have in common is not being validated against any empirical evidence.

Multiple copies of lexically, semantically, or functionally similar source code may be referred to as *reused code*, *duplicate code* or as a *clone*.<sup>iv</sup>

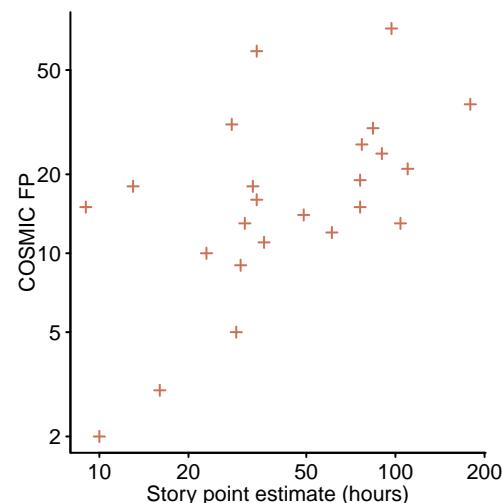


Figure 5.16: Estimated effort to implement 24 story-points and corresponding COSMIC function. Data from Commeyne et al.<sup>317</sup> [code](#)

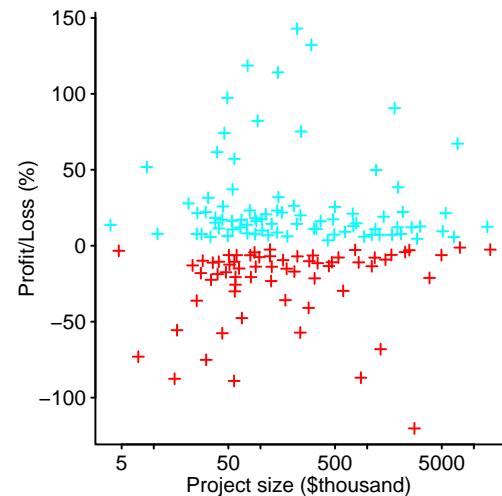


Figure 5.17: Percentage profit/loss on 145 fixed-price software development contracts. Data extracted from Coombs.<sup>326</sup> [code](#)

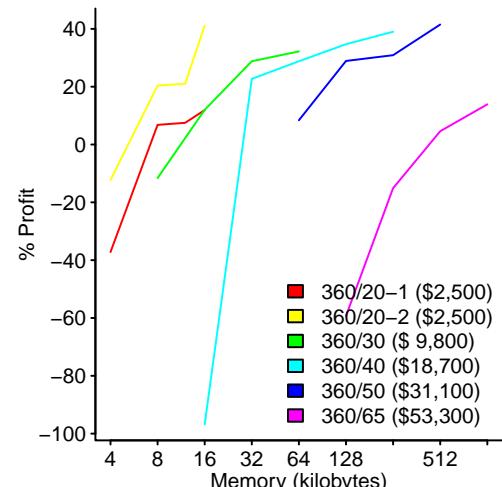


Figure 5.18: IBM's profit margin on all System 360s sold in 1966, by system memory capacity in kilobytes; monthly rental cost during 1967 in parentheses. Data from DeLaarter.<sup>390</sup> [code](#)

<sup>iv</sup> Clone is a term commonly used in academic papers.

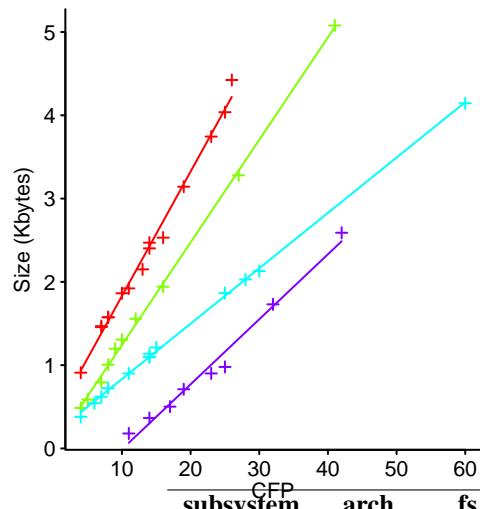


Figure 5.19: COSMIC function-points and compiled size (in kilobytes) of components in four different ECUs; lines show fitted regression model. Data from Lind et al.<sup>971</sup> code kernel 3.0 1.8 2.6 2.2

subsystem	arch	fs	kernel	mm	net	sound	drivers	crypto	others	LOC
MIPS function-points and compiled size components in four different ECUs fitted regression model. Data from Lind kernel	3.0	1.8	7.9	0.6	2.3	1.6	2.8	0.1	0.8	30,629
mm	2.6	2.2	0.8	6.2	1.7	1.1	2.0	0.0	0.7	23,490
net	1.8	2.5	1.1	0.7	20.7	2.1	3.7	0.1	1.0	334,325
sound	2.3	2.0	1.0	0.6	2.2	27.4	4.6	0.2	1.1	373,109
drivers	2.3	1.7	0.6	0.4	1.8	2.0	21.4	0.1	0.6	2,344,594
crypto	2.3	2.2	0.3	0.1	1.1	1.5	2.5	26.1	2.2	9,157
others	3.8	1.9	0.8	0.4	1.7	1.5	2.6	0.3	15.2	49,016

Table 5.2 Percentage of a module's source code cloned within and across subsystems of Linux 2.6.6. Data from Li et al. [95].

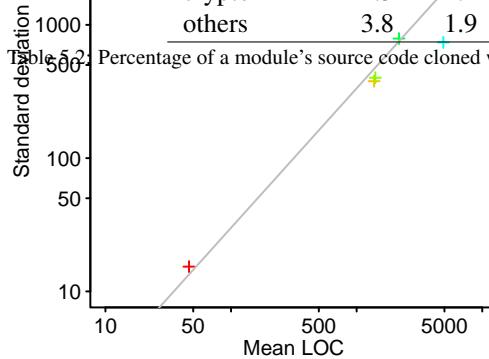


Figure 5.20: Mean LOC against standard deviation of LOC, for multiple implementations of seven distinct problems; grey line is fitted regression. Data from: Andalib et al.,<sup>40</sup> Jørgensen,<sup>798</sup> Lauterbach,<sup>929</sup> McAllister et al.,<sup>1033</sup> Selby et al.,<sup>1406</sup> Shimasaki et al.,<sup>1432</sup> van der Meulen.<sup>1591</sup>

Are clones consistently propagated during maintenance<sup>1554</sup> ... have data...

Source code exists to implement some functionality, it either has to be written or adapted from code that found elsewhere. There have been a series of papers investigating whether cloned code is more fault prone than non-cloned code.<sup>724</sup> in general these have failed to fully control for faults in non-cloned versions of the code. There are specific faults patterns that are the result of copy-and-paste errors,<sup>137</sup> and these are discussed in Chapter 6.

emailed twice (no reply...)?

Reasons for not reusing code include the cost of performing due diligence to ensure that property rights are respected (clones of code appearing on Stack Overflow<sup>v</sup> have been found in Android Apps<sup>38</sup> having incompatible licenses), ego (e.g., being recognized as the author of functionality), and hedonism (people enjoy inventing their own wheel and will argue against using somebody else's).

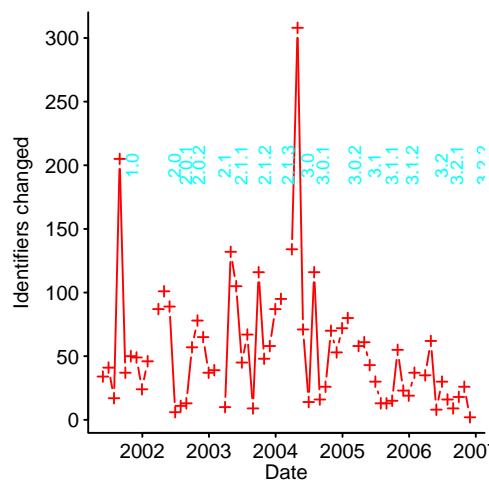


Figure 5.21: Number of identifiers renamed, each month, in the source of Eclipse-JDT; version released on given date shown. Data from Eshkevari et al.<sup>457</sup> code

## 5.3 Pitching for projects

Bidding on tenders, convincing senior management to fund a project, selling the benefits of a bespoke solution to potential clients, all involve making commitments that directly impact project implementation. An appreciation of some of the choices that might be made while pitching for a project may be useful for understanding why things are the way they are.

A client interested in acquiring bespoke software may quickly change their mind, if they hear a development cost that is much higher than expected, or a delivery date much later than wanted. Optimal frog boiling entails starting low and increasing at a rate that does not disturb...

If it has been decided that a project will be implemented internally, within the company, then company politics decides who does what.

<sup>v</sup> Example code on Stack Overflow is governed by a Creative Commons Attribution-ShareAlike 3.0 Unported license.

Companies and government departments needing bespoke software may put together a list of requirements, along perhaps with a specification, and invite interested parties to submit bids to implement the system.

Companies in the business of developing bespoke software need to maintain a pipeline of projects, opening with client qualification and closing with contract signature.<sup>1465</sup> Acquiring paying project work is the responsibility of the sales department and is outside the scope of this book.

Many of the factors involved in project bidding are likely to be common to engineering projects in general, e.g., highway procurement.<sup>94</sup> Bidding decisions can be driven by factors that have little or no connection with the technical aspects of software implementation, or its costs; some of these include:

- probability of being successful on bids for other projects. If work is currently scarce, it may be worthwhile accepting a small profit margin, or even none at all, simply to have work that keeps the business running. A software development organization, whether it contains one person or thousands, needs to schedule its activities to keep everyone busy and the money flowing in.

Some of the schedule estimates in Figure 5.2 can be explained by companies assigning developers to more than one project at the same time, improving staff efficiency by having something else for them to do when they are blocked from working on their primary project,

- bid what the maximum price the client is currently willing to pay. A profitable strategy when the client estimate is significantly greater than the actual; if the client perceives a project to be important enough, they are likely to be willing to pay more once existing monies are spent. It is better to pay £2 million for a system that provides a good return on investment than the £1 million actually budgeted, if the lower prices system is not worth having,
- the likely value of estimates submitted by other bidders; competition is a hard task master,
- bidding low to win and ensure that wording in the contract allows for increases due to unanticipated work. Prior experience shows that clients often want to change the requirements, estimating the cost of the new requirements occurs after the competitive bidding process (see Figure 3.18). The report by the Queensland health payroll system commission of inquiry<sup>285</sup> offers some insight into this approach. Clients often prefer to continue to work with a supplier who has run into, even substantial, difficulties,<sup>188</sup>
- bidding low to win, planning to make substantial profits during the maintenance phase. This strategy is based on having a reasonable expectation that the client will use the software for many years and that the software will need substantial maintenance,
- bidding on projects that are small, relative to the size of the development company, as a means of getting a foot in the door to gain access to work involving larger projects, e.g., becoming an approved supplier.

A study by Jørgensen and Carelius investigated the impact of changes in the project specification on the amount bid; estimators in group A made an estimate based on a one-page specification and then a second estimate based on an eleven-page specification; estimators in group B made an estimate based on the eleven-page specification only. Figure 5.22 shows bids made by two groups of estimators from the same company; for additional analysis, see [projects/proj-bidding.R](#).

Estimates will be affected by the characteristics of cognitive processing of numeric values (see Section 2.7.2), e.g., the measurement units used (such monthly or yearly)<sup>7</sup> can lead to different answers being given.

The bidding on a project for a new IT system for Magistrates' Courts (the Libra project),<sup>188</sup> started with ICL submitting a bid of £146 million, when it became public there was only one bidder this was increased to £184 million over 10.5 years, a contract was signed, a revised contract was renegotiated for £319 million, ICL threatened to repudiate the renegotiated contract and proposed a new price of £400 million, reduced its proposed price to £384 million and after agreement could not be reached signed a revised contract for £232 million over 8.5 years.

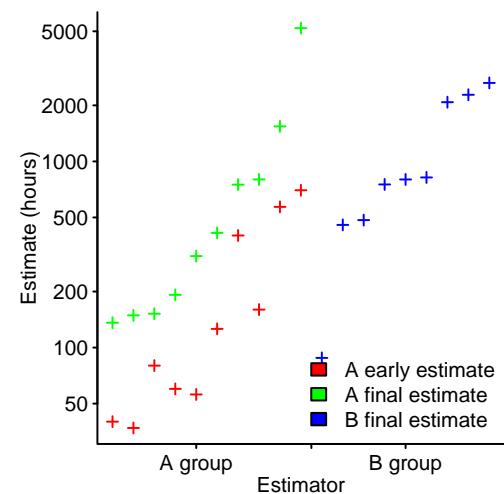


Figure 5.22: Bids made by 19 estimators from the same company (divided into two groups for the experiment). Data from Jørgensen et al.<sup>800</sup> code

### 5.3.1 Contracts

A signed contract is the starting point of serious investment of resources in project implementation (some projects never involve a contract, and work may start on a project before a contract is signed). Having to read a contract after it has been signed is an indication that one of the parties involved is not happy; nobody wants to have to deal with lawyers.<sup>1038</sup>

What is the payment schedule for the project? The two primary contract types are *fixed price* and *time and materials* (also known as *cost plus*; where the client pays the vendor costs plus an agreed percentage profit. e.g., a margin of 10-15%<sup>318</sup>).

On projects of any size, agreed amounts are paid at specified milestones. Milestones are a way for the client to monitor progress, and the vendor to receive some income for the work they have done. The use of milestones favours a sequential development viewpoint and one study<sup>64</sup> found that the waterfall model is effectively written into contracts.

From the client's perspective a fixed price contract appears attractive, but from the vendor's perspective this type of contract may be unacceptably risky (one study<sup>596</sup> found that vendors preferred a fixed price contract when they could increase their profit margin by leveraging particular staff expertise; another study<sup>956</sup> found that fixed-price was only used for trusted vendors). Writing a sufficiently exact specification of what the system is expected to do, along with tightly defined acceptance criteria is time-consuming and costly, which means the contract has to contain a mechanism to handle changes to the requirements; such a mechanism that is open to exploitation by both clients and vendors.

A time and materials contract has the advantage that vendors are willing to accept open-ended requirements, but has the disadvantage (to the client) that the vendor has no incentive to keep costs down.

Contracts sometimes include penalty clauses and incentive fees (which are meaningless unless linked to performance targets<sup>447</sup>).

if client cancels a project, the vendor contract should have been written to ensure that a profit is made on the work done... Depending on reason for cancellation, to what extent does reputation may suffer...

A study by Webster<sup>1646</sup> analysed legal disputes involving system failure filed in the period 1976-2000 (120 were found). The cases could be broadly divided into seven categories: client claims the installed system is defective in some way and vendor fails to repair it, installed system does not live up to the claims made by the vendor, a project starts, and the date of final delivery continues to slip and remain in the future, unplanned obsolescence (client discovers that the system either no longer meet its needs or that the vendor will no longer support it), the vendor changes the functionality or configuration of the system resulting in unpleasant for one or more client, a three-way tangle between vendor, leasing company and client, miscellaneous.

Many commercial transactions are governed by standard form contracts. A study by Marotta-Wurgler<sup>1020</sup> analysed 647 software license agreements from various markets; almost all had a net bias, relative to relevant default rules, in favor of the software company (who wrote the agreement).

The issue of personal integrity is discussed in [?].

Require rights to the source code, have to ensure that developers do not make inappropriate use of code (e.g., downloaded from the Internet<sup>1466</sup>) having incompatible licenses...

?

Client and vendor are in an asymmetric information situation... moral hazard...

?

could extract data...?

emailed, could extract...?, ?, ?

emailed for data...?

## 5.4 Paths to delivery

A project to deliver a working system involves researching, in detail, what the client wants (i.e., requirements gathering), formulating a design<sup>346</sup> that will do this, implementing it, and testing to ensure an acceptable level of confidence in its behavior (testing is covered in Chapter 6);<sup>1117</sup> for successful systems, deployment to customers marks the start of another project, ongoing maintenance and upgrades.

Managing the implementation a software system is an exercise in juggling the available resources, managing the risks (e.g., understanding what is most likely to go wrong<sup>79</sup> and having some idea about what to do about it), along with keeping the client convinced<sup>vi</sup> that a working system will be delivered within budget and schedule; these are all standard project management issues.<sup>862</sup> Managements interest in technical issues focuses on the amount of resource they are likely to consume, and the developer skill-set needed to implement them.

Progress reports detailing poor progress and increased costs may be ignored,<sup>830</sup> or trigger an escalation of commitment<sup>vii</sup>...

?

?

Some of the risks include the client changing the requirements<sup>vii</sup>, proposed solutions, to problems that have not been solved before (at least by the people available to work on the project) not working as planned, the discovery of new problems and changes to the known problems generating costs and schedule delays, and the hardware not being capable of doing what is required...

Developers and clients can very different views about the risks involved.<sup>900</sup>

Figure 5.23 shows an implementation schedule for one of the projects studied by Ge and Xu.<sup>553</sup> This schedule is based on breaking the project into 19 components, with some component dependencies known in advance along with the skills needed to implement them, information on the expected availability of developers having given skills was enough to create a schedule; as the implementation progresses the information is uncovered may result in schedule changes, e.g., more components dependencies, and available resources changes (such as developers ceasing to be available). This plot gives the impression of a project under control, in practice slippage on another project can prevent scheduled staff being available, a requirement change can generate a new dependency that prevents a subcomponent being implemented concurrently; schedules evolve, sometimes beyond all recognition.

A study by Yu<sup>1690</sup> investigated the development of three multi-million pound projects. The chronology of events documented gives some insight into how responsibility for the implementation of functionality, and associated costs, can shift between project contractors as new issues are uncovered and contracted budgets are spent. Figure 5.24 shows the changes in contractors' project cost estimates, for their own work, over the duration of the project.

A project has a beginning, an ending and for non-trivial projects milestones in-between. Every person who works on a project has their own beginning and ending.

When does a project start? For one person, work on a project starts when they first think of bidding on a contract, writing a proposal or starting to spend time thinking about an idea. Developers might not even know about the project they are about to work on until a contract has been signed, and they are allocated to work on it.

When does a project end? Depending on the kind of project being developed, it might finish when the client makes the final contract payment, a product is first offered for sale, or the code is no longer considered to be a release candidate; once the system is being used a shift to a maintenance mode of working may feel like a continuation, under another name, for some. Cancellation is an ending that nobody wants.

Requirements, as perceived by the client, change, leading to the inevitable request for the project to support an updated list of requirements.

Vendors want to keep the client happy, but not if it means losing a significant amount of money. A good client relationship manager will know when to tell the client that extra money

<sup>vi</sup> Successful politicians focus on being elected and then staying in office;<sup>368</sup> successful managers focus on getting projects and then keeping the client on-board; unconvincing clients cancel projects, or look for others to take it over.

<sup>vii</sup> A well written contract includes provisions for the extra costs and schedule slippage.

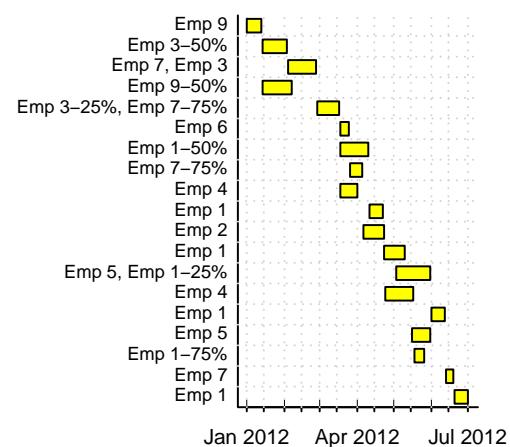


Figure 5.23: Initial implementation schedule, with employee number(s) given for each task (percentage given when not 100%) for a project. Data from Ge et al.<sup>553</sup> [code](#)

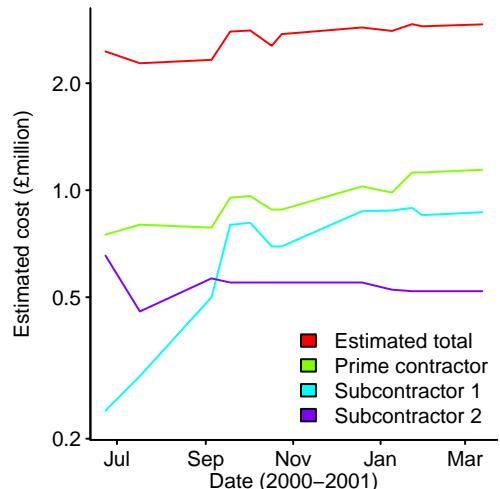


Figure 5.24: Estimated cost of developing a bespoke software system by the three companies contracted to do the work. Data from Yu.<sup>1690</sup> [code](#)

is needed to support the requested change and when to accept it without charging (any connection with implementation effort is only guaranteed to occur for change requests having significant cost impact)...

Project implementation strategy is strongly influenced by the cost impact of changes to requirements; the risk cost profile of changes...

When there is a very high cost associated with failing to detect an important requirement, it is worth making a significant investment in having a high degree of confidence that all the necessary requirements are known, e.g., see Figure 5.33.

Which requirements drive implementation costs and how likely are they to change? Brand<sup>200</sup> introduced the concept of *shearing layers* to refer to the way buildings contain multiple layers of change (Lim<sup>964</sup> categorised the RALIC requirements into five layers)...

A study by Simcoe<sup>1441</sup> investigated the modularity of communication protocol standards involved in supporting the Internet. Figure 5.25 shows the number of citations from IETF and W3C Standard documents, grouped by protocol layer, that reference Standard documents in the same and other layers. Treating citations as a proxy for dependencies, 89% are along the main diagonal (a uniform distribution would produce 17%), a value that might be used to estimate the extent to which dependencies will impact concurrent implementation of multiple protocols; dependencies discovered during implementation may substantially change the initial estimate.

Existing requirements may be modified on the basis of what the code does, rather than what the specification said it should do...<sup>227</sup>

Agreement between *clients* about whether requirements have been met...<sup>1278</sup>

The issue of ambiguity in requirement specifications is discussed in Chapter 6.

Gause and Weinberg<sup>550</sup> provide a readable tour through the various aspects of requirements handling in industry.

When a new product is being developed, there are benefits to first writing the User's manual and then treating this as the requirements specification<sup>153</sup>...

Use cases... Aimed towards specifying what a system is supposed to do... extract data from paper?

list of features removed from Linux... [feature-removal-schedule.txt](#)

Daily client contact has been found to have an impact on final project effort, see Figure 11.43. The study did not collect any data on the extent to which the planned project goals were achieved, and it is possible that daily contact enabled the contractor to convince the client to be willing to accept a deliverable that did not support the functionality originally agreed upon...

#### 5.4.1 Development methodologies

The difference between projects in established engineering disciplines (e.g., bridge building in Civil engineering) and projects in disciplines where empirically established practices have not yet been established (e.g., software development) is that in the former projects are mainly a problem of how to make the best use of known alternatives, while the latter involves discovering what the workable alternatives might be (e.g., learning by trying ideas until something that works well enough is found).

The first development methodology,<sup>587</sup> written in 1947, specified a six-step development process, starting with those who performed the high-level conceptual activities (i.e., the scientists and engineers) and ending with those who performed what was considered to be the straightforward coding activity (the coders). Over time the complexity of the task resulted in a transfer implementation authority and power to those doing the coding.<sup>448</sup>

The *Waterfall model* continues to haunt software project management, despite repeated exorcisms over several decades.<sup>921</sup> The paper<sup>1357</sup> that gave birth to this demon meme contained a diagram, with accompanying text claiming this approach was risky and invited failure, with the diagrams and text on subsequent pages showing the recommended approach to producing the desired outcome. The how not to do it approach, illustrated in the first diagram, was taken up, becoming known as the waterfall model and included in the first version of an influential

standard, DoD-Std-2167<sup>393</sup> as the recommended project management technique.<sup>viii</sup> Projects have been implemented using a Waterfall approach,<sup>1675</sup> but it is not considered to be the approach most likely to succeed. data...

The U.S. National Defense Authorization Act, for fiscal year 2010, specified that an iterative acquisition process be used for information technology systems; Section 804:<sup>994</sup> contains the headline requirements “(B) multiple, rapidly executed increments or releases of capability; (C) early, successive prototyping to support an evolutionary approach; . . .”. However, the wording used for the implementation of the new process specifies: “. . . well-scoped and well-defined requirements.”, i.e., bureaucratic inertia holds sway.

The battle over the communication protocols used to implement the Internet is perhaps the largest example of a waterfall (the OSI seven-layer model, ISO 7498 or ITU-T X.200, documented the requirements first, which vendors could then implement) vs. iterative approach (the IETF process was/is based on rough consensus and running code<sup>1364</sup>).

The economic environment and power structure in which a project is developed can have a significant impact on the management techniques used. For instance, the documentation and cost/schedule oversight involved in large government contracts requires a methodology capable of generating the necessary cost, schedule and documents; in winner take-all markets there is a strong incentive to be actively engaging with customers as soon as possible, and a methodology capable of producing regularly updated working systems provides a means of rapidly responding to customer demands as well as reducing the delay between writing code and generating revenue from it.

A study by Rico,<sup>1330</sup> going back over the 50-years, identified 32 major techniques (broken down by hardware era and market conditions) that it was claimed would produce savings in development or maintenance. There have been a few field studies of the methodologies actually used (rather than what managers claim to be using); system development methodologies have been found to provide management support for necessary fictions,<sup>1134</sup> e.g., a means to creating an image of control to the client or others outside the development group.

Iterative development has been independently discovered many times.<sup>921</sup> The advantages of iterative development include: not requiring a large upfront investment in requirements gathering, the potential to reduce wasted effort implementing unwanted requirements (through feedback received from users of the current system), working software makes it possible to start building a customer base (a crucial requirement in winner take-all markets), and the lag between writing code and earning income from it is reduced.

The additional cost paid for using iterative development, compared to an upfront design approach, is having to constantly reengineer code, to adapt it to support functionality that it was not originally designed to support.

If the cost of modifying the design is very high, it may be cost effective to do as much design as possible before writing code; the cost of design changes may be high when: hardware is involved, many organizations are involved and working to one design...

When customer requirements are rapidly changing, or contain many unknowns, the design may get out-of-date quickly, and it may be more cost effective to actively drive the design by ship something working (e.g., evolving series of languages, and their corresponding compilers;<sup>111</sup> see [projects/J04.R](#)). User feedback is the input to each iteration, and without appropriate feedback iterative projects can fail.<sup>609</sup>

The Internet provides an ideal ecosystem for the use of iterative techniques. During the growth of Internet usage the requirements were uncertain and subject to rapid change, many market niches had winner-take-all characteristics, and the Internet provided a distribution channel for frequent software updates...

The extent to which techniques used to implement previous projects are applicable to the current project... If the exact project has not been implemented before, then projects containing very similar features can be used as a guide.

Test driven development (TDD) emailed for more (available data is poor)...?

Pair programming... emailed...?

---

<sup>viii</sup> Subsequent versions removed this recommendation and more recent versions recommend incremental and iterative development. The primary author of DoD-Std-2167 expressed regret for creating the strict waterfall-based standard, that others advised him it was an excellent model; in hindsight, had he known about incremental and iterative development, this would have been strongly recommended, rather than the waterfall mode.<sup>921</sup>

Survey data...? Bayesian Statistics in Software Engineering<sup>531</sup> ... version-one/

A study by Özbek<sup>1202</sup> investigated attempts to introduce software engineering innovations into 13 open source projects in a one-year period. Of the 83 innovations identified in the respective project email discussions 30 (36.1%) were successful, 37 (44.6%) failed and 16 (19.3%) classified as unknown.

?

### 5.4.2 Managing progress

How is the project progressing towards the target, in cost and time, when a project deliverable is accepted by the client?

A project involves work being done and people doing it. Work and staff have to be meshed together in a schedule; project managers will have a view on what has to be optimized. A company employing a relatively fixed number of developers may seek to schedule work so that employee always have something productive to do, while a company with a contract to deliver a system by a given date will seek to schedule staff to optimise for delivery dates.

Scheduling a software project involves solving the kinds of problems encountered on many other kinds of creation projects, e.g., staff availability (in time and skills), dependencies between work items<sup>194</sup> and other projects, and hardware resources. Software specific issues include the difficulty of finding people with the necessary skills and high turnover rate of current staff. Project staffing is discussed in Section 5.5.

Tracking progress... Earned value management...

Analyzing project progress data in isolation may be misleading. A study by Curtis, Sheppard and Kruesi<sup>347</sup> investigated the characteristics (e.g., effort and faults found) of the implementation of five relatively independent subcomponents of one system; Figure 5.26 shows how effort, in person hours, changed as the projects progressed. While the five subcomponents were implemented by different teams, it is not known whether teams members worked on other projects within their company. Figure 5.5 shows how, at a multi-project level, total available effort may not significantly change, but large changes can occur at the single project level (because higher management optimise staffing across projects).

?

While it may be obvious to those working on a project that the schedule will not be met, nobody wants to be the bearer of bad news and management rarely have anything to gain by reporting bad news earlier than they have to.

Once the client believes the estimated completion date and/or cost is not going to be met, the project is either scaled back, cancelled, or a new completion estimate is accepted. Scheduling when to tell clients about project delivery slippage and/or a potential cost overrun is an essential project management skill.

The uncertainty around the schedule of a successful project is likely to decrease as it progresses towards completion. The metaphor of a *cone of uncertainty* is sometimes used for analysing project uncertainty; in practice this metaphor is at best useless. Schedule uncertainty is asymmetric, with underestimates dominating, the x-axis is calculated using a quantity that unknown until project completion (i.e., the duration of the project) and curved boundary seen in plots are a mathematical artefact. Figure 5.27 shows a plot of project percentage actually completed against the ratio  $\frac{\text{Actual}}{\text{Estimated}}$  for the corresponding project (4.6% of estimate completion dates are less than the actual date); the curved boundary does not contain any information, it is purely a mathematical artefact created by the choice of axis, i.e., the following holds:

$$\frac{\text{Actual}}{\text{Estimated}} \leq x_{\text{percentage}} \cdot \text{Actual}$$

cancelling *Actual* gives:  $\frac{1}{\text{Estimated}} \leq x_{\text{percentage}}$ ; whatever the value of *Estimated*, its point always appears below or on the  $\frac{1}{x}$  curve in the plot.

A study by Little<sup>973</sup> investigated schedule estimation for 121 projects at Landmark Graphics between 1999 and 2002; an estimated release date was made at the start and whenever the core team reached consensus on a new date (average 7.2 estimates per project). The extent to which the schedule estimation characteristics found in Landmark projects, for the period

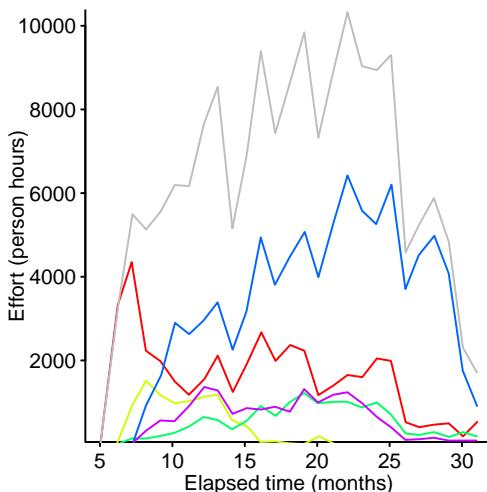


Figure 5.26: Effort, in person hours per month, used in the implementation of the five components making up the PAVE PAWS project (grey line shows total effort). Data extracted from Curtis et al.<sup>347</sup> [code](#)

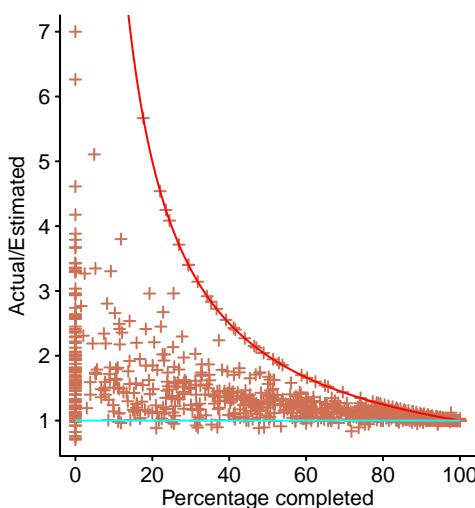


Figure 5.27: Percentage of actual project duration elapsed when 882 schedule estimates were made, during 121 projects, against estimated/actual time ratio (boundary maximum in red). Data kindly provided by Little.<sup>973</sup> [code](#)

of the data, will depend on corporate culture and requirements volatility. The Landmark Graphics corporate culture viewed estimates as targets, i.e., “what’s the earliest date by which you can’t prove you won’t be finished?”,<sup>973</sup> different schedule characteristics are likely to be found in a corporate culture that punishes the bearer of bad news.

Figure 5.28 shows the number of release date estimates made for 121 projects, for a corresponding initial estimated project duration (on x-axis).

Reasons for failing to meet a project deadline include (all starting points for lawyers looking for the best strategy to use, to argue their client’s case after a project fails<sup>1344</sup>): an unrealistic schedule, a failure of project management, changes in the requirements, encountering unexpected problems, staffing problems (i.e., recruiting or retaining people with the required skills) and being blocked because a dependency is not available.<sup>194</sup> Missed deadlines are common, and the usual response is to release an updated schedule; after too many missed deadlines projects are cancelled... data...

When is a new estimate for a project’s release date most likely to occur? If a lot of effort was invested in analyzing a project before it started, then new information, relevant to delivery date, is less likely to be discovered at the start of a project, compared to when little upfront analysis has been made. As the estimated delivery date approaches, it becomes easier for team members to judge the likelihood of delivery occurring.

If management want to continue development of a project after it has failed to be delivered by the scheduled date, they have to ask for the necessary resources; those paying for the work have to be given sufficient notice about the expected need for more resources, so that those resources can be made available (or not).

Figure 5.29 shows 882 changed delivery date announcement, with percentage elapsed time when the estimate was announced (based on the current estimated duration of the project) along the x-axis, and percentage change in project duration (for the corresponding project) along the y-axis (red line is a loess fit). The large changes in duration tend to occur near the start of projects, with smaller changes towards the estimated end date.

The blue line is a density plot of the percentage elapsed time when a schedule change was announced (density values not shown). The flurry of new estimates making large changes the delivery date suggest that a lot of new information is discovered during project startup. Over 50% of all estimates are made in the last 71% of estimated remaining time and 25% in the last 6.4% of remaining time.

Parkinson’s law claims that work expands to fill the time available. When management gives an estimated duration for a project, it is possible that those involved choose to work in a way that meets the estimate (assuming it would have been possible to complete the project in less time).

A study by van Oorschot, Bertrand and Rutte<sup>1594</sup> investigated the completion time of 424 work packages involving advanced micro-lithography systems, each having an estimated lead time. Figure 5.30 shows the number of work packages completed within a given amount of time, with colored lines showing stratification by estimated lead time.

emailed... eventually perhaps...?

Using an iterative process... time-boxed, fixed time periods, between each release...

Development groups following an Agile methodology make frequent releases, with each containing a small incremental update (Agile is an umbrella term applied to a variety of iterative and incremental software development methodologies).

Shirking, free-riding... Data promised...?

### 5.4.3 Major activities

The major activities involved in producing a system include requirements, design, implementation (or coding), testing and deployment. These activities have a natural ordering in the sense that it is unwise to deploy without some degree of testing, which requires code to test, which ideally has been designed and implemented a customer requirement.

These project activities are often called project phases, implying both an ordering, and a distinct period during which one activity is performed.

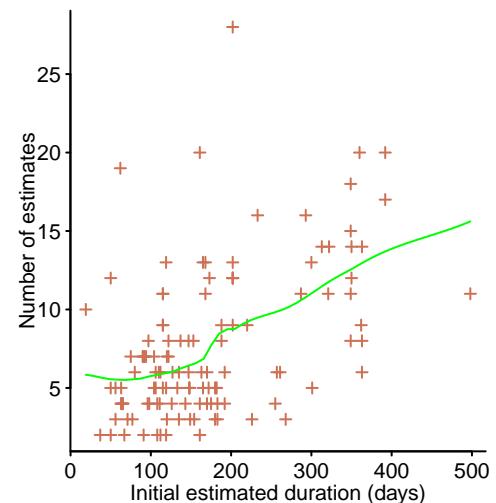


Figure 5.28: Initial estimated project duration against number of schedule estimates made before completion, for 121 projects; line is a loess fit. Data kindly provided by Little.<sup>973</sup> [code](#)

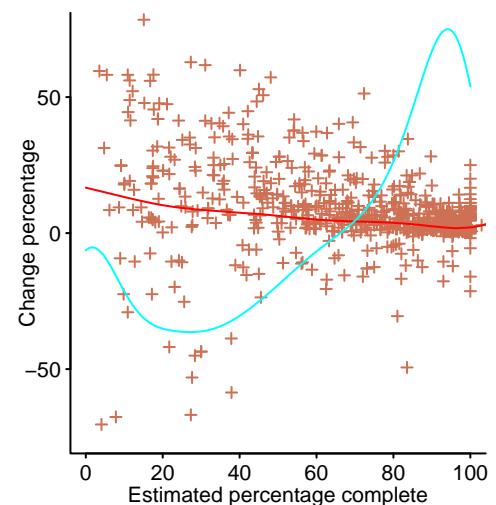


Figure 5.29: Percentage change in 882 estimated delivery dates announced at a given percentage of the estimated elapsed time of the corresponding project, for 121 projects (red is a loess fit); blue line is a density plot of percentage estimated duration when estimate made. Data kindly provided by Little.<sup>973</sup> [code](#)

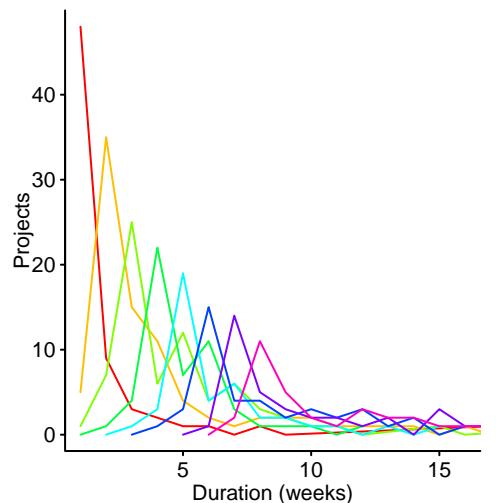
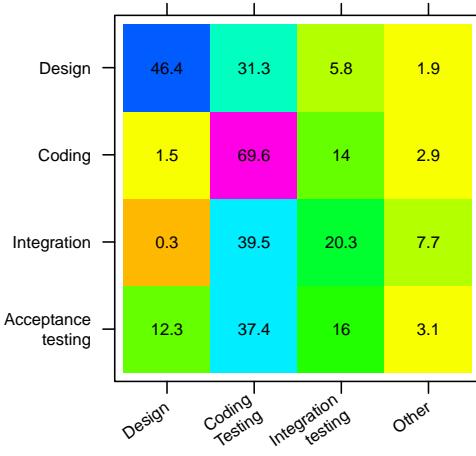


Figure 5.30: Number of work packages completed within a given time; colored lines are work packages having the same estimated lead time. Data extracted from van Oorschot et al.<sup>1594</sup> [code](#)

A study by Zelkowitz<sup>1697</sup> investigated when work from a particular activity was performed, relative to other activities (for thirteen projects, average total effort 13,522 hours). Figure 5.31 shows considerable overlap between all activity, e.g., 31.3% of the time spent on design occurred during the coding and testing activity.

also see [projects/zelkowitz-effect.R...](#)



Phase work overlapped with

Figure 5.31: Phase during which work on a given activity of development was actually performed, average percentages over 13 projects. Data from Zelkowitz.<sup>1697</sup> [code](#)

The extent to which documentation is a major activity, or a minor after-thought, depends on the client...

How are the available resources distributed across the major project activities?

A study by Condon, Regardie, Stark and Waligora<sup>321</sup> investigated 39 applications developed by the NASA Flight Dynamics Division between 1976 and 1992; Figure 5.32 shows ternary plots of the percentage effort, in time (red), and percentage schedule, in elapsed time (blue), for design, coding and testing (mean percentages for the three activities were: effort time: 24%, 43 and 33 respectively and schedule elapsed time: 33%, 34 and 33).

Figure 5.33 shows a ternary plot for design/coding/test effort for projects developed by a computer manufacturer (red), telecoms company (green), space projects (blue) and major defense systems (purple).

There is a clustering of effort breakdowns for different application areas; the mean percentage design, coding and testing effort were: computer/telecoms (17%, 57, 26) and Space/Defence (36%, 20, 43). There is less scope to recover from the failures of software systems operating in some space/defense environments; this usage characteristic makes greater investment in design and testing worthwhile.

#### 5.4.4 Discovering functionality needed for acceptance

Requirements are the supply chain of software development...

What functionality does a software system need to support for a project delivery to be accepted<sup>ix</sup> by the client?

The *requirements gathering* process (other terms used to describe this process include: *requirements elicitation*, *requirements capture* and *systems analysis*) will be influenced by the environment in which the project is being developed:

- when an existing way of working, or system, is being replaced (over half the projects in one recent survey<sup>799</sup>), the stakeholders of the existing system are an important source of requirements information,
- when the software is to be sold to multiple customers, as a product, those in charge of the project select the requirements. In this entrepreneurial role, the trade-off is minimising investment in implementing functionality against maximising sales income,
- when there is huge uncertainty, perhaps the client does not have enough information to decide between options or there are questions about technically feasibility, building a prototype can be a cost-effective method for evaluating ideas and technologies, helping decide and refine requirements.<sup>1464</sup>
- when starting an open source project the clients are those willing to do the work, or contribute funding.

?

The probability that a requirement will be added or changed, some aspect of the design has to be changed (because it is discovered that the proposed solution is unlikely to be viable), some code has to be rewritten, or the hardware has to be modified...

A study by Felici<sup>485</sup> analysed the evolution of requirements, over 22 releases, for eight features contained in the software of a safety-critical avionics system. Figure 5.34 shows the requirements for some features completely changing between releases, while the requirements for other features were unchanged over many releases.

<sup>ix</sup> Acceptance here means paying all the money specified in the contract, plus any subsequent agreed payments.

The relative cost, for different phases of development, of adding or changing a requirement, changing the design, rewriting code or modifying the hardware<sup>785</sup>...

How much effort might be required to produce a detailed requirements specification? One effort estimate<sup>785</sup> was 50 person years to produce, the C90 standard, a 219-page A4 document; the effort for the next version of the standard, C99 (a 538-page document), was estimated to be 12 person years.

What is a cost effective way of discovering requirements and their relative priority?

One possible source of requirements is the intended users of the software.

A *stakeholder* is someone who gains or loses something (e.g., status, money, change of job description) as a result of a project going live. Stakeholders are a source of political support, resistance to change<sup>1621</sup> and active subversion<sup>1353</sup> (i.e., doing what they can to obstruct progress on the project), may be able to provide essential requirements' information, or may be an explicit target for exclusion (e.g., criminals with an interest in security card access systems).

Requirements gathering is an iterative process<sup>279</sup>...

Failure to identify the important stakeholders can result in missing or poorly prioritized requirements, which can have a significant impact on project success. What impact do different stakeholder selection strategies have?

A study by Lim<sup>964</sup> investigated the University College London (UCL) project to combine different access control mechanisms into one, e.g., access to the library and fitness centre. The Replacement Access, Library and ID Card (RALIC) project had been deployed two years before the study started, and had more than 60 stakeholder groups. The project documentation, along with interviews of those involved (gathering data after project completion means some degree of hindsight bias will be present), was used to create the *Ground truth* of project stakeholder identification (85 people), their rank within a role, requirements and relative priorities.

[data/require.tgz](#)... [data/RALIC\\_datasets.zip](#)

The following two algorithms were used to create a final list of stakeholders:

- starting from an initial list of 22 names and 28 stakeholder roles, four iterations of Snowball sampling resulted in a total of 61 responses containing 127 stakeholder names+priorities and 70 stakeholder roles (known as the *Open list*),
- a list of 50 possible stakeholders was created from the project documentation. The people on this list were asked to indicate, which of those names on the list they considered to be stakeholders, and to assign them a salience<sup>x</sup> between 1 and 10, they were also given the option to suggest other names as possible stakeholders. This process generated a list containing 76 stakeholders names+priorities and 39 stakeholder roles (known as the *Closed list*).

How might a list of people, and the salience each of them assigns to others, be combined to a single salience value for each person?

Existing stakeholders are in a relationship network. Lim proposed the hypothesis that the rank of stakeholder roles calculated using social network metrics would be strongly correlated with the rank ordering of stakeholder roles in the Grounded truth list. Figure 5.35 shows the Open and Closed stakeholder salience values calculated using Pagerank (treating each stakeholder as a node in the network created using the respective lists; Pagerank was chosen for this example because it had one of the strongest correlations with the Ground truth ranking). Also, see [projects/requirements/stake-ground-cor.R](#).

Identifying stakeholders and future users is just the beginning. Once found they have to be convinced to commit time to a project they may have no immediate interest in; stakeholders will have their own motivations for specifying requirements. When the desired information is contained in the heads of a few key players, these need to be kept interested and involved throughout the project. Few people are practiced in requirements' specification and so obtaining the desired information is likely to be an iterative process, e.g., they describe solutions rather than requirements.

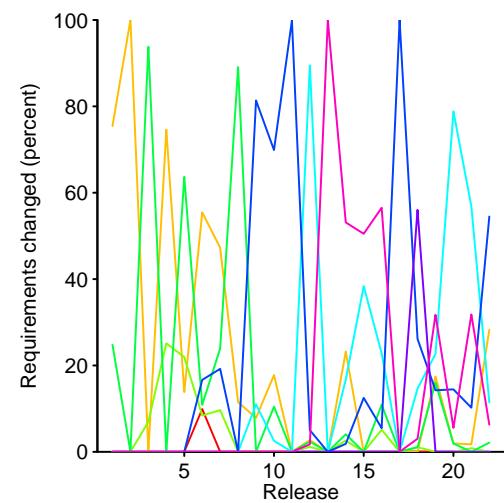


Figure 5.34: Percentage of requirements added/deleted/modified in eight features (colored lines) of a product over 22 releases. Data extracted from Felici.<sup>485</sup> [code](#)

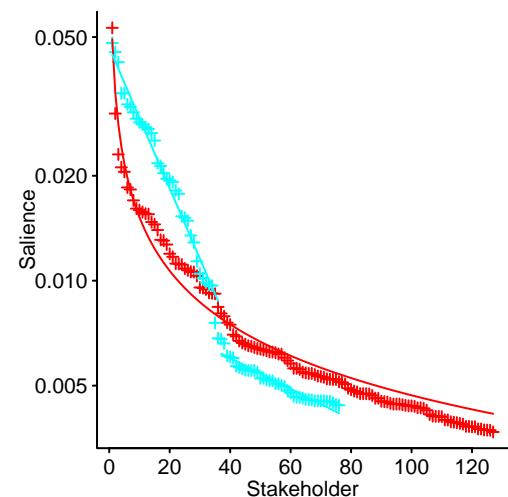


Figure 5.35: Pagerank of the stakeholders in the network created from the Open (red) and Closed (blue) stakeholder responses (values for each have been sorted). Data from Lim.<sup>964</sup> [code](#)

<sup>x</sup> The term *salience* is used to denote the level of a stakeholder's influence.

**Prioritization** The priority of some requirements will be ranked higher than others; concentrating resources on the high priority requirements is a cost-effective way of keeping the client happy and hopefully producing a more effective system (for the resources invested). Techniques for prioritising requirements include:

- an aggregated priority list: this involves averaging stakeholders' list of assigned values, possibly weighting each stakeholders' views.

To what extent are the final requirements' priorities dependent on one stakeholder? Calculating an average, with each stakeholder excluded in turn, is one method of estimating the variance of priority assignments.

A study by Regnell, Höst, Dag, Beremark and Hjel<sup>1319</sup> asked each stakeholder/subject to assign a value to every item on the list of requirements, without exceeding a specified maximum amount (i.e., to act as-if they had a fixed amount of money to spend on listed requirements). Figure 5.36 shows the average value assigned to each requirement, and the standard deviation in this value when stakeholders were excluded, one at a time.

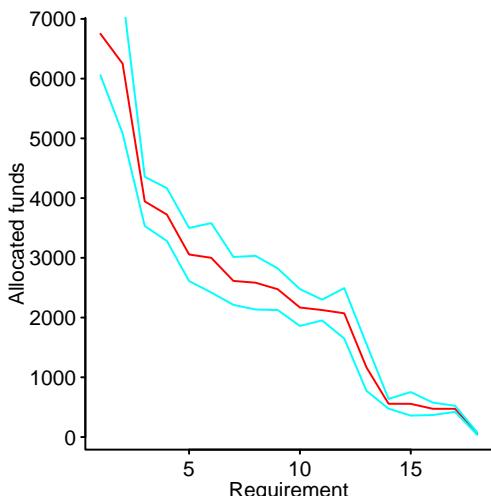


Figure 5.36: Average value assigned to requirements (red) and one standard deviation bounds (blue) based on omitting one stakeholder's priority value list. Data from Regnell et al.<sup>1319</sup> [code](#)

- performing a cost/benefit analysis on each requirement and prioritizing based on the benefits provided for a given cost<sup>826</sup> (see [projects/requirements/cost-value.R](#)),
- the *analytic hierarchy process* (AHP)... data...

The client who authorises the cheques has the final say on which requirements have to be implemented and their relative priority. The clients' primary project role is ensuring that their desired requirements are met; clients who fail to manage the requirements process end up spending their money on a bespoke system meeting somebody else's needs.<sup>1275</sup>

## 5.4.5 Implementation

Code is an exact specification of behavior. The behaviors that matter are those that have a client visible effect, which means developers have to make many invisible decisions, e.g., which sort routine to use. Section 6.4.2 discusses patterns of source usage. Mistakes are made during implementation and Chapter 6 discusses testing and reliability estimation.

How might project progress be measured? Lines of code is a poor metric because it is easily manipulated... i.e., Goodhart's law applies.

Percentage of time spent on activities during development... meetings, learning, coding, blocked... data...

What are the characteristics of the ongoing production of a software system developed using an iterative process?

7digital<sup>1</sup> is a digital media delivery company that uses an Agile process to develop an open API platform providing business to business digital media services; 3,238 features were implemented by the development team between April 2009 and July 2012.

The start/done dates represent elapsed time, not total development time (i.e., work on a feature may have been stalled for a time). The measurements were not used for control or evaluation of developers, and there was no incentive for developers to change their behavior based on being measured, i.e., Goodhart's law does not apply.

Figure 5.37 shows the average time taken to implement a feature, over time (smoothed using a 25-day rolling average).

The variance in average implementation time has a change-point around 650 days (a change-point in the mean occurs around 780 days). This change suggests that one or more significant changes occurred as the second anniversary approached.

Figure 5.38 shows the number of features requiring a given number of elapsed working days for their implementation (upper first 650-days, lower post 650-days); a zero-truncated negative binomial distribution is a good fit to both sets of data.

Having a good fit to the same distribution for both the pre and post 650-day datasets suggests that the change in behavior was not a fundamental change, but akin to turning a dial on the

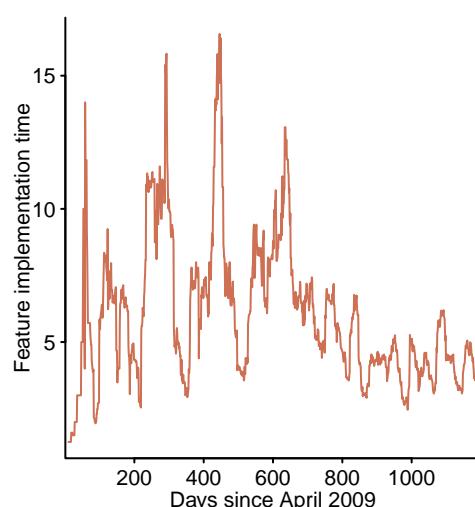


Figure 5.37: Average number of days taken to implement a feature, over time; smoothed using a 25-day rolling mean. Data kindly supplied by 7Digital.<sup>1</sup> [code](#)

distribution parameters one way or the other (the first 650-days has a greater mean and variance than post 650-days). If the two sets of data were better fitted by different distributions, the processes generating the two patterns of behavior are likely to have been very different.

A Negative binomial distribution suggests that the feature implementation processes has many factors involved (i.e., many values, each drawn from a different Poisson distribution, with the variation having a Gamma distribution).

Figure 5.39 shows the number of new features and number of bug fixes started per day (smoothed using a 25-day rolling mean).

7digital is a growing company, during the recording period the number of developers grew from 14 to 35, the size of its code base and number of customers is not available. The number of developers who worked on each feature was not recorded and while developers write the software, it is clients who often report most of the bugs; client information is not present in the dataset.

Possible causes for the continuing increase in new feature starts include an increase in the number of developers and/or existing developers becoming more skilled in breaking work down into smaller features (i.e., feature implementation time stays about the same because fewer developers are working on each feature, making developers available to start on new features).

Both the number of bugs and non-bug features has trended upwards, and the ratio is well below one, i.e., they spend more effort adding features than fixing bugs. Some of the increase has been generated by the significant increase in number of developers over the time period, and it is also possible the group has become better at dividing work into smaller feature work items or that having implemented the basic core of the products less effort is now needed to create new features.

Work on feature implementation is very unlikely to always finish at the end of a day. Can the fit be improved by adjusting for measurement quantization?

Fitting 1,000 randomly modified half-day measurements and averaging over all results shows that the fit is slightly worse than the original data (as measured by various goodness of fit criteria): see [projects/agile-work/feat-half-day-7dig.R](#).

System dynamics model for Agile development...?

Number of merges against commits/team size/branches... broken merges...?

Work-talk patterns...??

???

What do developers spend their time doing? ...20% of time spent on build issues, 11% of time spent maintaining build... Startup cost or one time change cost for the build...?

## 5.4.6 Documentation

The term *documentation* might be applied to anything, requirements, specifications, code documentation, comments in code, testing procedures, bug reports and user manuals; source code is sometimes referred to as its own documentation or as self-documenting.

Q/A sites... accuracy of answers, correct answers that are not correct for later product releases...

Incentives for writing documentation include:

- a costly display signalling commitment, e.g., management wants everybody to believe the project will be around for a long time, or is important,
- an investment intended to provide a worthwhile return by reducing the time/cost of learning for future project hires... and the data for this...
- fulfil a contract requirement. This requirement may have appeared in other contracts used by the customer, and nobody is willing to remove it; perhaps customer management believes that while they do not understand code, they will understand prose,

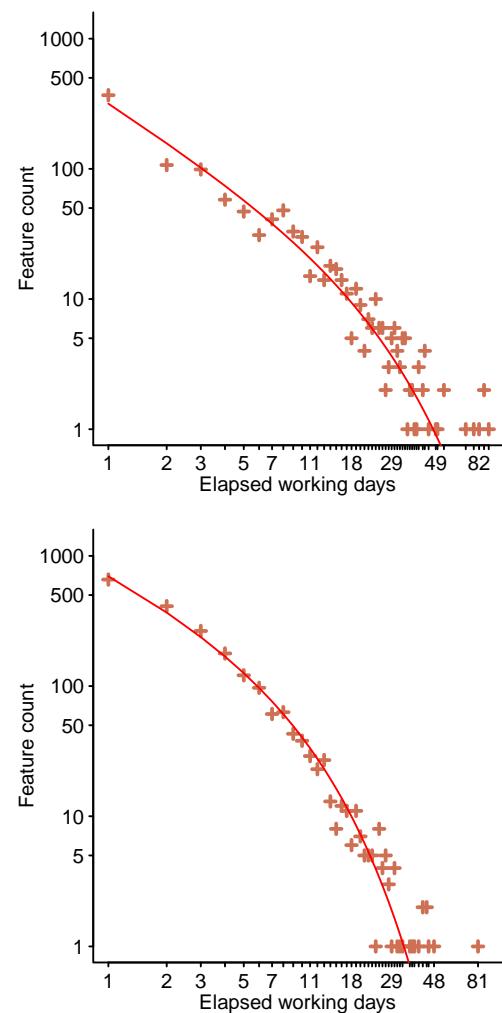


Figure 5.38: Number of features whose implementation took a given number of elapsed workdays; upper first 650-days, lower post 650-days. Fitted zero-truncated negative binomial distribution in green. Data kindly supplied by 7Digital.<sup>1</sup> [code](#)

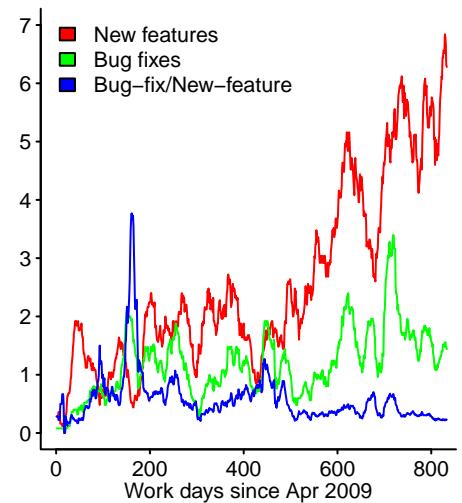


Figure 5.39: Number of feature developments started on a given work day (red new features, green bugs fixes, blue ratio of two values; 25-day rolling mean). Data kindly supplied by 7Digital.<sup>1</sup> [code](#)

Development projects that derive income from consultancy and training, have an incentive to minimise the publically available documentation. Knowledge of how a software system works may be the basis for a saleable service, and availability of documentation reduces this commercial potential.

The existence of documentation can be a liability, in that courts have considered vendors liable for breach of warranty when differences exist between product behavior, and the behavior specified in documentation.<sup>821</sup>

How much are documents changed over time, what gets changed over time... Figure 4.58 shows... links in source code become dead<sup>659</sup>...

Issues around interpreting the contents of documentation are covered in Chapter 6...

### 5.4.7 Deployment

New software systems often go through a series of staged releases (e.g., alpha, then beta releases and release candidates); the intent is to uncover any unexpected problems... data on average time in various stages or release...

A successful project is likely to make multiple releases, with the number and time-frame of releases driven by the cost of making one and the likely income from users...

Traditionally the next release of a system has been either time-based (i.e., specified as occurring on a given date) or feature based (i.e., when a specific set of features has been implemented).

Projects using time-based releases need to ensure that the system is in a usable state on the release date. One approach to creating a usable release is to pause work on new features sometime before the next release, switching development effort to creating a usable release (e.g., performing extensive testing); the term *code freeze* is sometimes used to describe the phase of development.

A study by Laukkanen, Paasivaara, Itkonen, Lassenius and Arvonen<sup>927</sup> investigated the number of commits prior to 19 releases of four components of a software system at Ericsson; the length of the code freeze phase varied between releases. Figure ?? shows that the pattern of commits per day, before and during code freeze, varied between releases... [projects/laukkanen2017/laukkanen2017.R](#).

A study by Zhao, Serebrenik, Zhou, Filkov and Vasilescu<sup>1706</sup> investigated the impact of switching to use Travis CI as part...

?

Iterative development has the potential to make continuous releases, but in practice fixed (e.g., weekly) or irregular intervals...?

?

?

For systems supporting many build-time configurations options, it may be more cost effective<sup>649</sup> to concentrate on the popular option combinations and wait until problems with other configurations are reported (rather than invest resources checking many options that will never be used).

While some bespoke software is targeted at particular computing hardware, long-lasting software may be ported to a variety of different platforms. The cost/benefit analysis of investing in reducing the cost of porting to a different platform (i.e., reducing the switching cost) requires knowing the likelihood of this event occurring.

A study by Peukert<sup>1240</sup> investigated the switching costs of outsourced IT systems, as experienced by U.S. Credit Unions. Figure 5.40 shows the survival curve of IT outsourcing suppliers employed by 2,382 Credit Unions, over the period 2000 to 2010.

Given sufficient customer demand, it is worthwhile for a vendor to continue maintaining a product. Adding new features to an existing product creates something that can be sold to existing customers and helps to compete with new market entrants.

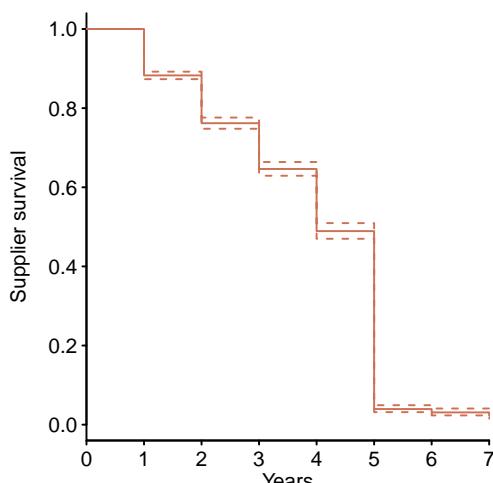


Figure 5.40: Survival curve of IT outsourcing suppliers continuing to work for 2,382 Credit Unions. Data kindly provided by Peukert.<sup>1240</sup> [code](#)

Figure 5.41 shows the growth of lines of code, command line options, words in the manual and messages supported by PC-Lint (a C/C++ static analysis tool), in the 11 major releases over 28 years.

A study by Hofman<sup>710</sup> investigated user evaluation of products... [developers/Hofman-exp1.R](#)

## 5.4.8 Refactoring

Refactoring is an investment activity that assumes there will be a need to modify the code again in the future, and it is more cost effective to restructure the code now, rather than at some future date. Possible reasons for time shifting an investment in code include developers not having alternative work to do, or an expectation that the unknown future modifications will need to be made quickly, and it is worth investing time now to reduce future development schedules; also, developers may feel peer pressure to produce code that follows accepted ecosystem norms (e.g., open source code that is about to be released)...

A study by Kawrykow and Robillard<sup>828</sup> of 24,000 change sets from seven long-lived Java programs found that between 3% and 16% of all method updates consisted entirely of non-essential modifications, e.g., renaming of local variables, and trivial keyword modifications.

A study by Eshkevari, Arnaoudova, Di Penta, Oliveto, Guénéuc and Antoniol<sup>457</sup> of identifier renamings in Eclipse-JDT and Tomcat, found that almost half of were applied to method names, a quarter to field names and most of the remaining to local variables and parameter names. No common patterns of grammatical form, of the renaming, were found (e.g., changing from a noun to a verb occurred in under 1% of cases). Figure 5.21 shows the number of identifiers renamed in each month, along with release dates; no correlation appears to exist between the number of identifiers renamed and releases.

## 5.5 Development teams

A project needs people to do the work, and the complexity of staff scheduling ranges from a selecting a fixed number of staff to work on a project until delivery, to balancing staffing needs across multiple large projects in multiple locations.

A study by Buettner<sup>227</sup> investigated large software intensive systems and included an analysis of various staffing related issues, including: staffing level over time (Figure 11.66) and staff work patterns (Figure 11.58).

Distribution of number of people in a team... data?... Selecting team members<sup>864</sup>...

When do team members come from?... existing in-house staff, hire developers for the duration of the project (e.g., contractors)...

Team size... [embedded\\_survey/](#)

<sup>361</sup> ...

Changing staffing requirements and changing skill required staff as project progresses...

What set of skills does a development team require and how might these skills be divided up among team members?...

When a project uses multiple programming languages, a team either needs to include developers familiar with multiple languages, or include additional developers to handle specific languages.

Figure 5.43 shows the number of different languages used in a sample of 100,000 GitHub projects (make was not counted as a language).

What is the best way to organize a software project team?

Drawing a parallel with the methods of production used in manufacturing factories, the factory concept for software projects<sup>348</sup> was proposed and used by several large companies. <sup>xi</sup> The perceived advantages of this approach are the same as those it provides to traditional manufacturers, e.g., control of the production process and reduction in the need for highly skilled employees.

<sup>xi</sup> It was particularly popular in Japan.<sup>349</sup> Your author has not been able to locate any data on companies recently using the factory concept to produce software.

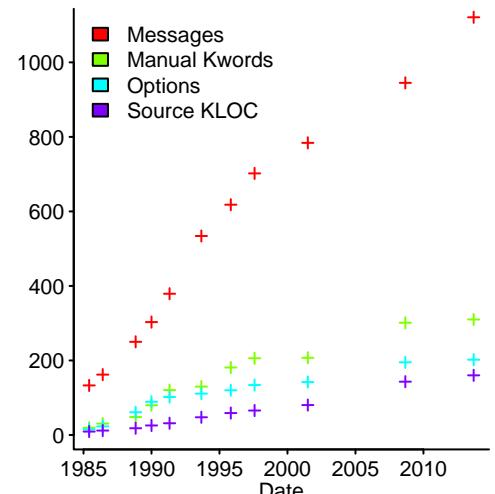


Figure 5.41: Growth, over 11 major releases in 28 years, of messages supported, command line options, kilo-words in product manual, and thousand lines of code in PC-Lint. Data kindly provided by Gimpel.<sup>572</sup> [code](#)

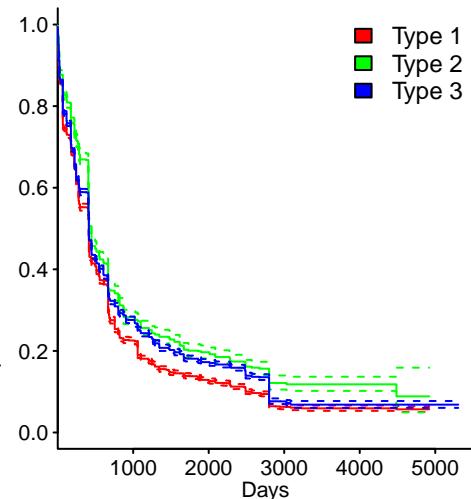


Figure 5.42: Survival curves of clones in the Linux high/medium/low level SCSI subsystems. Data from Wang.<sup>1631</sup> [code](#)

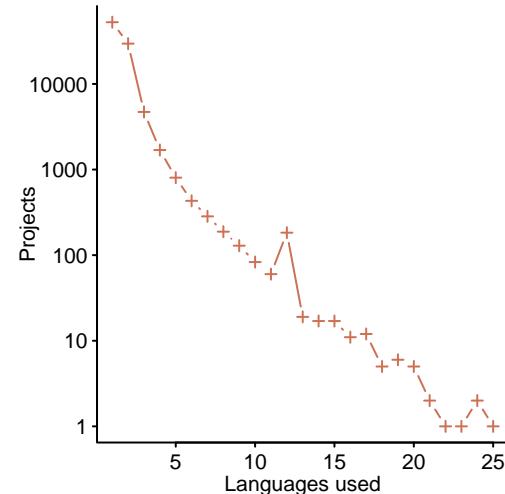


Figure 5.43: Number of projects making use of a given number of different languages in a sample of 100,000 GitHub projects. Data kindly provided by Bissyande.<sup>167</sup> [code](#)

There have been few experimental comparisons<sup>1670</sup> of the many project development techniques that have been proposed over the years.

The chief programmer team<sup>96</sup> approach to team organization was designed to handle environments where many of the available programmers are inexperienced (a common situation in a rapidly growing field); an experienced developer is appointed as the chief programmer and is responsible for doing the heavy lifting and allocating the tasks requiring less skill to others. This form of team organization dates from the late 1960s, when programming involved a lot of clerical activity and in its original formulation emphasis is placed on delegating this clerical activity,

Team members' awareness of other members of the team...?

Self-organising teams is one approach to dividing up the available manpower...

Half-life of project team members, developer turnover means that x% of code has been written by people no longer working on the project...

??

?

Application domain knowledge<sup>1410</sup>...?

?

### 5.5.1 New staff

New people may join a project as part of planned growth, the need to handle new work, existing people leaving, management wanting to reduce the dependency on a few critical people, and many other reasons.

Project member turnover... joining/leaving data...

Training people (e.g., developers, documentation writers) who are new to a project reduces the amount of effort available to building the system in the short term. Training is an investment in people whose benefit is the post-training productivity these people bring to a project.

Brooks' Law<sup>219</sup> says: "Adding manpower to a late software project makes it later", but does not say anything about the impact of not adding manpower to a late project. Under what conditions does adding a person to a project cause it to be delayed?

If we assume a new person diverts, from the project they join, a total effort,  $T_e$ , in training and that after  $D_t$  units of time the trained person contributes  $E_n$  effort per unit time until the project deadline; unless the following inequality holds, training a new person results in the project being delayed:

$$E_{a1}D_r < (E_{a1}D_t - T_e) + (E_{a2} + E_n)(D_r - D_t)$$

where:  $E_{a1}$  is the total daily effort produced by the team before the addition of a new person,  $E_{a2}$  the total daily effort produced by the original team after the addition, and  $D_r$  is the number of units of time between the start of training, and the delivery date/time.

Adding a person to a team can both reduce the productivity of the original team (e.g., by increasing the inter-person communication overhead) and increase their productivity (e.g., by providing a skill that means the whole is greater than the sum of its parts). Assuming that  $E_{a2} = cE_{a1}$ , the equation simplifies to:

$$T_e < (D_r - D_t)(E_n - (1 - c)E_{a1})$$

and...

In practice a new person's effort contribution ramps up from zero, perhaps even during the training period, to a relatively constant long term daily average;  $E_n$ ...

The effort,  $E_T$ , that has to be invested in training a new project member will depend on their existing level of expertise with the application domain, tools being used, coding skills, etc (pretty much everything was new, back in the day, for the project analysed by Brooks, so  $E_T$  was probably very high). There is also the important ability, or lack of, to pick things up quickly, i.e., their learning rate...

If many people are being added to a project at the same time, it is easy to imagine it grinding to a halt because of all the minor congestion that occurs within the network of dependencies that project progress is waiting on.

In practice new developers have to learn directly from the source code.... Risks of out-of-date documentation...

Splitting activities between communicating and everything else (e.g., producing code), how large can a team become before total team output begins to decline when more people are added?

Assuming the communications overhead, for each team member, is given by:  $t_0(D^\alpha - 1)$ , where  $t_0$  is the percentage of one person's time spent communicating in a two-person team,  $D$  the number of people in the team and  $\alpha$  a constant greater than zero. The maximum team size, before adding people starts reducing total output, is given by:<sup>1538</sup>

$$D_{max} = \left[ \frac{1+t_0}{(1+\alpha)t_0} \right]^{\frac{1}{\alpha}}$$

If  $\alpha = 1$  (i.e., everybody on the project incurs the same communications overhead), then  $D_{max} = \frac{1+t_0}{2t_0}$ , which for small  $t_0$  is:  $D_{max} \approx \frac{1}{2t_0}$ . For example, if team members spend 10% of their time communicating with every other team member:  $D_{max} = \frac{1+0.1}{2 \times 0.1} \approx 5$ .

In this team of five, 50% of each person's time is spent communicating.

$$\text{If } \alpha = 0.8, \text{ we have: } D_{max} = \left[ \frac{1+0.1}{(1+0.8) \times 0.1} \right]^{\frac{1}{0.8}} \approx 10.$$

Figure 5.44 shows...

If people spend most of their time communicating with a few people and very little with the other team members, the distribution of communication time may have an exponential distribution; the (normalised) communications overhead is:  $1 - e^{-(D-1)t_1}$ , where  $t_1$  is a constant found by fitting data from the two-person team (before any more people are added to the team).

The maximum team size is now:

$$D_{max} = \frac{1}{t_1}$$

and, if  $t_1 = 0.1$ , then:  $D_{max} = \frac{1}{0.1} = 10$ .

In this team of ten, 63% of each person's time is spent communicating (team size can be bigger, but each member will spend more time communicating compared to the linear overhead case).

Data on distribution of team member communication time...

?

[projects/Effectsofgroupsize\\_Peltokorpi.R](#)

## 5.6 Product updates

Once operational, software systems are subject to the economics of do nothing, update or replace.

A successful software system may attract funding (e.g., paying users) that supports more development work. The, so-called, maintenance of a software system is a (potentially long term) project in its own right.

Chapter 4 discusses one reason for continuing to invest in developing commercial products, i.e., there is a potential source of revenue after the initial sale, and the investment is a signal to potential customers that the product is not dead; see [ecosystems/maint-dev-ratio.R](#).

A company developing bespoke software for in-house may want the software to be adapted, as the world in which it is used changes. Developers are needed to fix reported problems and provide support. What else is there to do with their time? Adding features is a hedonic incentive for developers to stay in their current job (the issue of attracting developers to perform maintenance work on existing software is discussed)... data?...

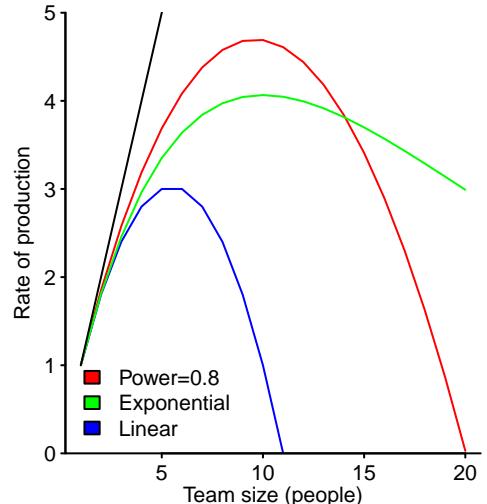


Figure 5.44: Production rate of a team containing a given number of people, with communication overhead  $t_0 = t_1 = 0.1$ , and various distributions of percentage communication time; black line is zero communications overhead.

A company that has been responsible for maintaining software for a customer is in the best position to estimate actual costs, and the price the customer is willing to continue paying for maintenance (see Figure 3.18). Without detailed maintenance cost information other companies are unlikely to be willing to bid to take over maintenance of an existing system<sup>162</sup> (at least unless the customer is willing to underwrite their risk).

What are the main activities that occur during software maintenance? Perhaps the most widely cited breakdown of activities are those listed in a paper<sup>962</sup> published 38 years ago (obtained by analysing the responses given by 69 maintenance managers answering a questionnaire containing 50 questions, not the most reliable way of obtaining information)... Nosek & Palvia questionnaire results from 1990... More recent studies<sup>1379</sup> have found very different results (which is to be expected given the small sample size used in the earlier work). Software is maintained in response to customer demand, and so the activities will be driven by this demand, e.g., very large systems in a relatively stable market<sup>1099</sup> will have a different change profile than smaller systems in sold into a rapidly changing market.

Once funding for an academic research project has ended, investment on the software written for that project may cease. A study<sup>731</sup> of 214 packages associated papers published between 2001-2015, in the journal Molecular Ecology Resources, found that 73% had not been updated since publication.

A study by Elliott<sup>441</sup> investigated the staffing levels for large commercial systems (2 MLOC) over the lifetime of the code (lifetime was defined as the time between a system containing the code first going live, and the same system being replaced; *renewal* was the terminology used by the author). The study did not look at staffing for the writing of new systems or maintenance of existing systems.

Figure 5.45 shows the average number of staff needed for renewal of code having a given average lifetime, along with a biexponential regression fit.

Software systems are changed via *patches* to existing code. In the case of the Linux kernel submitted patches first have to pass a review process; patches that pass review then have to be accepted by the maintainer of the appropriate subsystem, these maintainers submit patches they consider worth including in the official kernel to Linus Torvalds (who maintains the official version).

A study by Dekleva<sup>389</sup> investigated the average monthly maintenance effort (in hours) spent on products developed using traditional and modern methods (from a 1992 perspective). Figure 5.46 shows the age of systems, and the corresponding time spent on monthly maintenance...

A study by Baysal, Kononenko, Holmes and Godfrey<sup>122</sup> tracked the progress of 34,535 patches submitted through the WebKit and Mozilla Firefox code review process between April 2011 and December 2012. Figure 5.47 shows the percentage of patches (as a percentage of submitted patches) being moved between various code review states in WebKit. TODO Blink data...

The extent to which developer fault handling review behavior depends on the company who submitted the patch... acceptance/rejection time for Apple/Google/other?

A study by Jiang, Adams and German<sup>775</sup> investigated attributes of the patch submission process, such as the time between submission and acceptance (around 30% of the patches that make it through review are accepted into the kernel); the data includes the 81,000+ patches to the kernel source, between 2005 and 2012. Project shows a kernel density plot of the interval between a patch passing review and being accepted by the appropriate subsystem maintainer, and the interval between a maintainer pushing a patch and it being accepted by Torvalds. Maintainers immediately accept half of patches that pass review (Torvalds 6%). The kernel is on roughly an 80 day (sd 12 days) release cycle; the rate at which Torvalds accepts patches steadily increases, before plummeting at the end of the cycle.

emailed for more data...?

?

To what extent might library dependencies be updated for new releases of a software system?...??

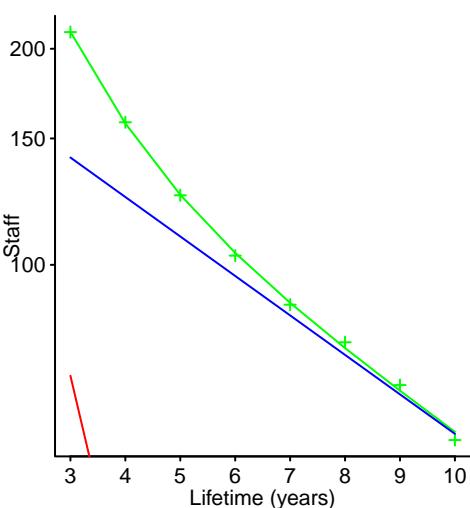


Figure 5.45: Average number of staff required to support renewal of code having a given average lifetime. Data extracted from Elliott.<sup>441</sup> code

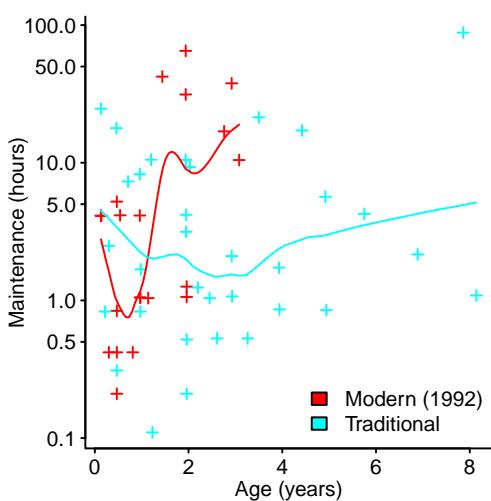


Figure 5.46: Age of systems, developed using one of two methodologies, and corresponding monthly maintenance time, along with loess fits. Data extracted from Dekleva.<sup>389</sup> code

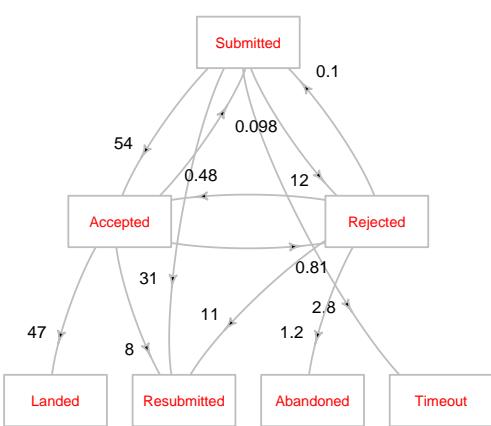


Figure 5.47: Percentage of patches submitted to WebKit (34,535 in total) transitioning between various stages of code review. Data from Baysal et al.<sup>122</sup> code

### 5.6.1 Project lifespan

Those involved in a project may decide to stop working on it...

The users of a software system may continue to use it after development on it has stopped...

?

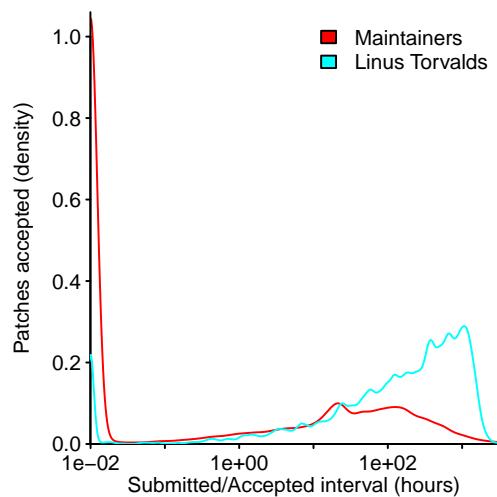


Figure 5.48: Density plot of interval between a patch passing review and being accepted by a maintainer, and interval between a maintainer pushing the patch to Linus Torvalds, and it being accepted into the blessed mainline (only patches accepted by Torvalds included). Data from Jiang et al.<sup>775</sup> code



# Chapter 6

## Reliability

### 6.1 Introduction

People are willing to continue using software which contains faults, that they sometimes experience,<sup>i</sup> provided it delivers a worthwhile benefit. The random walk of life can often be nudged to avoid unpleasantness, or the operational usage time limited to keep within acceptable safety limits.<sup>214</sup> Regions of acceptability may exist in programs containing many apparently major mistakes, supporting useful functionality.<sup>1334</sup>

Software systems containing likely fault experiences are shipped because it is not economically worthwhile fixing some of the mistakes made during their implementation; also, the process of finding and fixing problems, prior to release, is often constrained by available manpower and marketing deadlines.

Software release decisions involve weighing whether the supported functionality provides enough benefit to be attractive to customers (i.e., they will spend money to use it), after factoring in likely costs arising from faults experienced by customers (i.e., from lost sales, dealing with customer complaints and possible fixing problems, and making available an updated version).

How many fault experiences will customers tolerate, before they are unwilling to use software; are some kinds of fault experiences more likely to be tolerated than others? Customer utility function... Willingness to pay is a commonly used measure of risk acceptability and *As Low As Reasonably Practicable* (ALARP) is a term often applied...

In the U.S., federal agencies are required to adhere to an Executive Order<sup>ii</sup> that specifies: “Regulatory action shall not be undertaken unless the potential benefits to society for the regulation outweigh the potential costs to society.”. In some cases the courts have required that environmental, social and moral factors be included in the cost equation.<sup>258</sup>

Developers make the coding mistakes that create potential fault experiences, and the environment in which the code executes provides the input that results in faults occurring (which may be experienced by the user). This chapter discusses the kinds of mistakes made, where they occur in the development process, methods used to locate them and techniques for estimating how many fault experiences can potentially occur. Issues around the selection of algorithms is outside the scope of this chapter; algorithmic reliability issues include accuracy<sup>392</sup> and stability of numerical algorithms,<sup>697</sup> and solutions include minimising the error in a dot product by normalizing the values being multiplied.<sup>442</sup>

People make mistakes;<sup>1286,1316</sup> economic considerations dictate how much is invested in reducing the probability that mistakes costly leading to costly fault experiences remain (either contained in delivered software systems or as a component of a larger system). The fact that programs often contained many mistakes was a surprise to the early computer developers,<sup>1666</sup> as it is for people new to programming.

Operating as an engineering discipline, does not in itself ensure that a constructed system has the desired level of reliability. There has been, roughly, a 30-year cycle for bridge failures;<sup>1239</sup>

<sup>i</sup> Experience is the operative word, a fault may occur and not be recognized as such.

<sup>ii</sup> President Reagan signed Executive Order 12291 in 1981, and subsequent presidents have issued Executive Orders essentially affirming this requirement.<sup>1522</sup>

new design techniques and materials are introduced, and growing confidence in their use leads to overstepping the limits of what can safely be constructed.

What constitutes reliability, in a given context, is driven by customer requirements, e.g., in some situations it may be more desirable to produce an inaccurate answer than no answer at all, while in other situations no answer is more desirable than an inaccurate one.

The early computers were very expensive to buy and operate, and much of the software written in the 1960s and 1970s was for large corporations or government bodies; the US Department of Defence took an active role in researching software reliability, and much of the early published research is based on the software development environments used by DOD and NASA projects during this period.

The approach to computer system reliability promoted by these sponsors of early research set the outlook for much of what has followed. Emphasis is often placed either on large projects that are expected to be maintained over many years or software operating in situations where the cost of failure is extremely high and there is very limited time, if any, to fix problems (e.g., Space shuttle missions<sup>593</sup>).

This chapter discusses reliability from a cost/benefit perspective; yes, mistakes have a cost, but these can be outweighed by the benefits of releasing the software containing them. As with the other chapters, the target audience for this material is the vendor of the software, not its users; it is possible for vendors to consider a project a success because it produced a good return on investment, but for the user to consider it to be a failure because of the problems they experienced using it.

The relative low cost of modifying existing software, compared to hardware, provides greater flexibility for trading-off upfront costs against the cost of making changes later (e.g., by reducing the amount of testing before release), knowing that if necessary it is often practical to provide updates later. For some vendors, the Internet provides an almost zero cost means of distribution.

Mistakes in software can have a practical benefit for some people, for instance authors of computer malware have used mistakes in CPU emulators to detect that their activity may be monitored<sup>1205</sup> (and therefore the malware should remain inactive).

Mistakes are not unique to software systems; a study<sup>1070</sup> of citations in research papers found an average error rate of 20%.

The creation of mathematics shares many similarities with writing software, and many mistakes are made in mathematics;<sup>1268</sup> mathematical notation is a language with rules specifying syntax and permissible transformations. The size, complexity and technicality of some mathematical proofs has raised questions about the ability of anybody to check whether they are correct, e.g., Mochizuki's proof of the *abc* conjecture,<sup>255</sup> and the Hales-Ferguson proof of the **Kepler Conjecture**.<sup>904</sup> Many important theorems don't have proofs, only sketches of proofs and outline arguments that are believed to be correct;<sup>1136</sup> the sketches provide evidence used by other mathematicians to decide whether they believe a theorem is true (a theorem may be true, even although mistakes are made in the claimed proofs).

Mathematical proof differs from software in that the proof of a theorem may contain mistakes, but the theorem may still be true. For instance, in 1899 Hilbert found that Euclid's Elements (published around 300 BC) contained mistakes;<sup>1636</sup> the theorems were true and Hilbert was able to write correct proofs. Once a theorem is believed to be true, mathematicians have no reason to check its proof.

The social processes involved, in the mathematics community coming to believe that a theorem is true, is still coming to terms with believing machine-checked proofs.<sup>1260</sup> The nature and role of proof in mathematics continues to be debated.<sup>691</sup>

Mistakes are much less likely to be found in mathematical proofs than software, because a lot of specialist knowledge is needed to check new theorems in specialised areas, but a clueless button pusher can experience a fault in software simply by running it; also, there are few people checking proofs, while software is being checked every time it is executed.

Proposals<sup>706</sup> that programming should strive to be more like mathematics are based on the misconception that the process of creating proofs in mathematics is less error prone than creating software.<sup>369</sup>

Software that has been both developed and checked using formal analysis tools may still contain mistakes;<sup>517,558</sup> there is also the possibility of mistakes and ambiguities in the tools

used to perform the verification, e.g., ML.<sup>810</sup> One study<sup>1110</sup> of a software system that had been formally provided to be correct, found at least two mistakes per thousand lines remained.

Software differs from hardware in that, not only does it not wear out, but the variability in the inputs it has to process can have a significant impact on fault experiences. In software, repeatedly performing the same operation with the same input is expected to produce the same behavior (while hardware is expected to eventually fail). The dependency of fault experiences on the input distribution means much of the hardware-based reliability theory is not applicable.

Fixing a reported fault is one step in a chain of events that may result in users of the software receiving an update.

Vendors who supply an operating system vary in the amount of control they exercise over the software version their customers can run. Apple maintains a tight grip over use of iOS, and directly supplies updates to customers cryptographically signed for a particular device (i.e., the software can only be installed on the device that downloaded it). Google supplies the latest version of Android to OEMs and has no control over what, if any, updates these OEMs supply to customers (who are also free to install versions from third-party suppliers). Microsoft sells Windows 10 through OEMs, but makes available security fixes and updates for direct download by customers.

Figure 6.1 shows some of the connections between participants in the Android ecosystem (number of each kind in brackets), and some edges are labeled with the number of known updates flowing between particular participants (from July 2011 to March 2016).

Experiments designed to uncover unreliability issues may fail to find any. This does not mean that they are rare, the reason for failing to find a problem may be lack of statistical power (i.e., the likelihood of finding an effect if one exists); this topic is discussed in Section 10.2.3.

?

### 6.1.1 It's not a fault, it's a feature

The classification of program behavior as a fault or a feature may depend on the person doing the classification (e.g., user or developer). For instance, software written to manage a parts inventory may not be able to add a new item once the number of items it contains equals 65,536; a feature/fault that users will not encounter until an attempt is made to add an item that would take number of parts in the inventory past this value.

One fault report study<sup>692</sup> found that many reported faults were actually requests for enhancement.

Is insufficient or excessive accuracy in the numeric values calculated by a program, a fault or feature?

Variations in the behavior of software between different releases, or running the same code on different hardware can be as large as the behavior effect the user is looking for; a potential major problem when medical diagnosis is involved<sup>631</sup> ...

In some application domains it may be necessary to wait a long time before data becomes available, that can be used to check the accuracy of results calculated by a program, e.g., long range weather forecasts. The developers of the Community Earth System Model compare the results from previous climate calculations to determine whether modified code produces statistically different results.<sup>95</sup>

The accuracy of calculated results may be specified in the requirements, or the developer writing the code may be the only person who gives any thought to the question. When calculations involve floating-point values, developers may choose to play safe and use variables declared to have types capable of representing greater accuracy than is required<sup>1362</sup> (leading to higher than necessary resource usage, <sup>613</sup> e.g., memory, time and battery power).

Effects of using fewer representation bits in a floating-point datatype<sup>1536</sup>

Hardware design choices can increase the difficulty of obtaining accurate numeric results, e.g., the choice of representation for numeric values. Cost of implementation<sup>339</sup> resulted in binary becoming the dominant hardware representation for numeric values, despite decimal being the dominant representation within business. While cost of implementation eventually decreased to a point where processors supporting both binary and decimal are generally available, much existing software is written from a binary perspective.

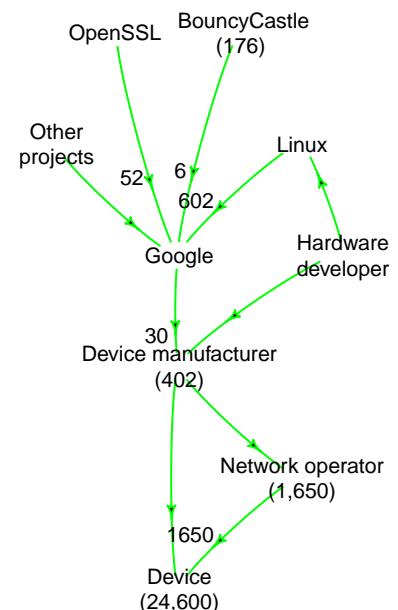


Figure 6.1: Flow of updates between participants in one Android ecosystem; number of each kind of member given in brackets, number of updates shipped on edges. Data from Thomas.<sup>1549</sup> code

?

### 6.1.2 Why do faults occur?

Two events are required for the person running an application to experience a software related fault:

- a mistake exists in the software, i.e., if the mistake is found, the developer who wrote the code would change it to behave as intended,
- the program processes input values that cause it to execute the code containing the mistake in a way that results in a fault being experienced<sup>336</sup> (software that is never used has no reported faults).

Some mistakes are more likely to be encountered than others because the required input values are more likely to occur during the use of the software. Any analysis of software reliability has to consider the interplay between the probabilistic nature of the input distribution and coding mistakes present in the source code (or configuration information).

An increase in the number of people using a program is likely to lead to an increase in reported faults, because of both an increase in possible reporters and an increase in the diversity of input values.

The Ultimate Debian Database project<sup>1578</sup> collects information about packages included in the Debian Linux distribution, from users who have opted-in to the Debian Popularity Contest. Figure 6.2 shows the numbers of installs (for the "wheezy" release) of a given packaged application against faults reported in that package and also age of the package against faults (data from the Debian Bug Tracking System, which is not the primary fault reporting system for some packages); also, see Figure 11.23.

A fitted regression model is, assuming a percentage measurement error:

$$\text{reported\_bugs} = e^{-0.15 + 0.17 \times \log(\text{insts}) + (30 + 2.3 \times \log(\text{insts})) \times \text{age} \times 10^{-5}}$$

For an *age* between 1,000—6,000 and installs between 10—20,000 (giving a  $\log(\text{insts})$  between 2—10), then the number of installations (a proxy for number of users) appears to play a larger role in the number of reported faults, compared to *age* (i.e., the amount of time the package has been included in the Debian distribution). There is a great deal of variance in the data...

A study<sup>1512</sup> of TCP checksum performance found that a far fewer corrupt network packets were detected in practice, than expected (by a factor of between 10 and 100). The difference between practice and expectation was caused by the non-uniform distribution of input values (the proof that checksum values are uniformly distributed assumes a uniform distribution of input values).

A hardware fault may cause the behavior of otherwise correctly behaving software to appear to be wrong, e.g., hardware is more likely to fail as the workload increases.<sup>763</sup>

### 6.1.3 Reported fault data

While reported faults have been studied almost since the start of software development, until open source bug repositories became available there was little publicly available fault data. The possibility of adverse publicity and fear of legal consequences of publishing information on problems found their software products was not lost on commercial organizations and nearly all of them treat such information as commercially confidential. While some companies maintained software fault databases,<sup>610</sup> these were not publicly available.

During the 1970s the Rome Air Defence Center published many detailed studies of software development,<sup>347</sup> and some included data on faults experienced in military projects.<sup>1672</sup> However, these reports were not widely known about or easy to obtain, until they became available via the Internet; a few were published as books.<sup>1547</sup>

The few pre-Open source datasets appearing in academic papers contained relatively few fault reports and if submitted for publication today would probably be rejected as not worthy

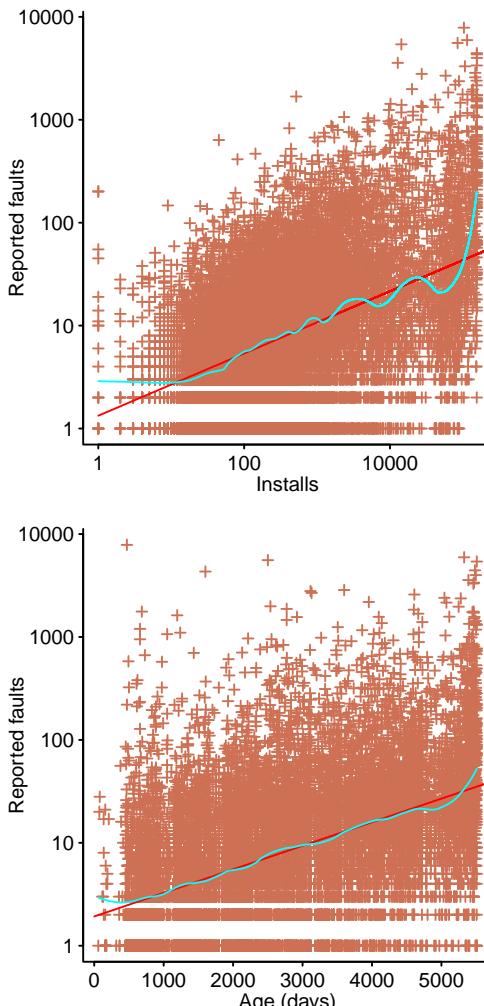


Figure 6.2: Reported faults against number of installations (upper) and age (lower). Data from the "wheezy" Debian release.<sup>1578</sup> code

of consideration. These studies<sup>1235, 1237</sup> usually investigated particular systems, listing percentages of faults found by development phase and the kinds of faults found; one study<sup>1628</sup> listed fault information relating to one application domain, medical device software. Knuth published<sup>870</sup> a list of all known mistakes in a widely used program (LaTex).

Confidentiality, unwillingness to share data, researchers not archiving their data once work on a project stops...

The economic impact of poor computer security has resulted in coding mistakes having security implications becoming a major category of interest. Several databases of such faults are actively maintained, including: The NVD (National Vulnerability Database<sup>1162</sup>), the VERIS Community Database (VCDB);<sup>1599</sup> an Open source vulnerability database, the Exploit database (EDB);<sup>430</sup> lists proof of concept vulnerabilities; mistakes in code which may be exploited to gain unauthorised access to a computer (vulnerabilities discovered by security researchers who have a motivation to show off their skills), and the Wooyun program<sup>1704</sup>...

A study by Sadat, Bener and Miransky<sup>y 1368</sup> investigated issues involving connecting duplicate fault reports. Figure 6.3 shows the connection graph for Eclipse report 6325 (report 4671 was the earliest report covering this issue).

How similar are the characteristics of Open source project fault report data compared with commercial fault data?

Various problems have been found with Open source fault report data, which is not to say that fault data on closed source projects is not without its own problems; these problems include:

- reported faults do not always appear in the fault report databases (e.g., serious bugs tend to be under-reported in commit logs;<sup>164</sup> a replication<sup>152</sup>). One study<sup>81</sup> of fault reports for Apache, over a 6-week period, found that only 48% of bug fixes were recorded as faults in the Bugzilla database; the working practice of the core developers was to discuss serious problems on the mailing list, and many fault experiences were never formally logged with Bugzilla,
- reported faults are misclassified. One study of reported faults (see Section 14.1.1) found that 42.6% of fault reports had been misclassified, with 39% of files marked as defective not actually containing any reported fault... requests for enhancement... Differences of opinion in how a reported fault should be classified<sup>1605</sup>... reported problem, on detailed investigation, turning out not to be a coding mistake,
- reporting bias: reported faults discovered through actively searching for them,<sup>470</sup> rather than normal program usage (e.g., Linux is a popular target for researchers, fuzzing tools; csmith generated source code targeted at compilers). The reporting of vulnerabilities contained, or not contained, in NVD has been found to be driven by a wide variety social, technical and economic pressures.<sup>292, 1153</sup> Data on faults discovered through an active search process may have different characteristics than faults experienced through normal program usage,
- reported faults are not contained within the source of the program cited, but is contained within a third-party library: a study by Ma, Chen, Zhang, Zhou and Xu<sup>995</sup> of the Python scientific package ecosystem found...
- reported fault could not be reproduced or was intermittent: a study<sup>1370</sup> of six servers found that on average 81% of the 266 reported faults analysed could be reproduced deterministically, 8% non-deterministically, 9% were timing dependent, plus various other cases,

A lot of fault analysis research is based on the faults reported against brand-name systems, such as Linux, Mozilla, Eclipse and NVD...

The fault report data does not always contain enough information to answer the questions being asked of it, e.g., using incidence data to distinguish between different exponential order fault growth models.<sup>1082</sup>

#### 6.1.4 Cultural outlook

Cultures vary in their attitude to the risk of personal injury and death. A study by Viscusi and Aldy<sup>1615</sup> investigated the value of a statistical life in 11 countries and found a range of

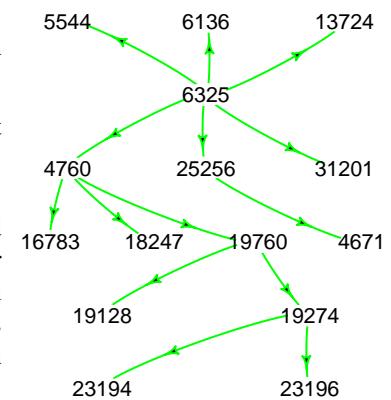


Figure 6.3: Duplicates of Eclipse fault report 4671 (report 6325 was finally chosen as the master report); arrows point to report marked as duplicate of an earlier report. Data from Sadat et al.<sup>1368</sup> code

estimates from \$0.7 million to \$20 million (adjusted to the dollar rate in 2000). Individuals in turn have their own perception of risk and sensitivity to the value of life.<sup>1462</sup> Some risks may be sufficiently outside of a persons' experience that they are willing to discount an occurrence affecting them, e.g., destruction of a city, country or all human life by a meteor impact.<sup>1468</sup>

Linguistic studies of concepts associated with the word *reliability* have been made for Japanese<sup>1088</sup> ...

A study by Budescu, Por, Broomell and Smithson<sup>226</sup> investigated how people in 24 countries, speaking 17 languages, interpreted uncertainty statements containing four probability terms (i.e., very unlikely, unlikely, likely and very likely, translated to the subjects' language). Figure 6.4 shows the mean percentage likelihood estimated by people in each country to statements containing each term.

Public perception of events influences and is influenced by media coverage (e.g., in a volcano vs. drought disaster, the drought needs to kill 40,000 times as many people as the volcano to achieve the same probability of media coverage<sup>437</sup>). A study<sup>437</sup> of disasters and media coverage found that when a disaster occurs while other stories are deemed more newsworthy, aid from U.S. disaster relief is less likely to occur.

Table 6.1, from a 2011 analysis by the UK Department for Transport,<sup>1559</sup> lists the average value that would have been saved, per casualty, had an accident not occurred.

Injury severity	Lost output	Human costs	Medical and ambulance	Total
Fatal	£545,040	£1,039,530	£940	£1,585,510
Serious	£21,000	£144,450	£12,720	£178,160
Slight	£2,220	£10,570	£940	£13,740
Average	£9,740	£35,740	£2,250	£47,470

Table 6.1: Average value of prevention per casualty, by severity and element of cost (human cost based on willingness-to-pay values); last line is average over all casualties. Data from UK Department for Transport.<sup>1559</sup>

A study by Costa and Kahn<sup>334</sup> investigated changes in the value of life in the USA, between 1940 and 1980. The range of estimates, adjusted to 1990 dollars, was \$713,000 to \$996,000 in 1940 and \$4.144 million to \$5.347 million in 1980.

Estimates of the number of deaths associated with computer related accidents contain a wide margin of error,<sup>1002</sup> with human-computer interaction being a commonly cited cause...

Governments are aware of the dangers of society becoming overly risk-averse, and some have published risk management policies<sup>1626</sup> ...

A report<sup>290</sup> from the UK's Department for Environment, Food and Rural Affairs provides an example of the kind of detailed analysis involved in calculating a monetary valuation of reducing risk.

The U.S. Department of Defense Standard MIL-STD-882E<sup>394</sup> defines and gives numbers for the terms, when applied to an individual item; the following are some of these terms:

- *Probable*: "Will occur several times in the life of an item"; probability of occurrence less than  $10^{-1}$  but greater than  $10^{-2}$ .
- *Remote*: "Unlikely, but possible to occur in the life of an item"; probability of occurrence less than  $10^{-3}$  but greater than  $10^{-6}$ .
- *Improbable*: "So unlikely, it can be assumed occurrence may not be experienced in the life of an item"; probability of occurrence less than  $10^{-6}$ .

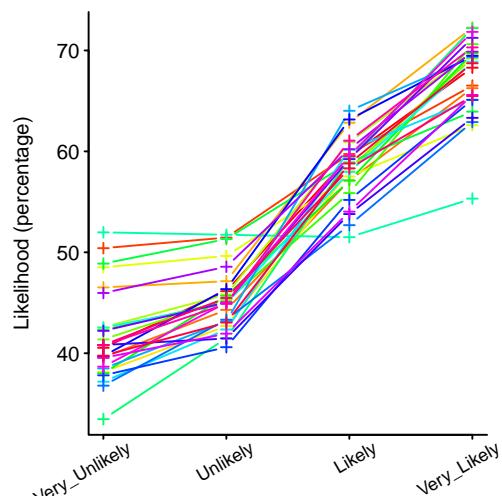


Figure 6.4: Mean percentage likelihood of (translated) statements containing a probabilistic term; one colored line per country. Data from Budescu et al.<sup>226</sup> code

Some government related organizations have published guidelines covering the use of software in various industries, e.g., in medical devices,<sup>477</sup> automotive<sup>1090, 1091</sup> ...

## 6.2 The search for profit

A vendor's approach to product reliability is driven by maximising return on investment in their chosen market.

There is often an asymmetry in the cost of implementation mistakes borne by the producer and consumer of software systems.

For single client development projects, the client is in a position to make trade-off decisions involving their estimated post-delivery fault experience costs and estimated pre-delivery mistake reduction costs (e.g., paying the vendor to do something).

For software written to be sold to multiple customers, the producer has to cover the costs for their mistakes (which may include costs arising from litigation<sup>822</sup>) and the customer...

The reputational cost, to the vendor, of customer fault experiences...

In some markets the likelihood of customer fault experiences is source of revenue to vendors, e.g., the profit margin on a maintenance contract....

In some markets, there may be a competitive advantage to being first to market with a new or updated product; the biggest component of fault experience costs may be lost sales rather than the cost of correcting mistakes later (i.e., take the risk that customers are not deterred by the greater number of faults experienced)... rapid development, *frontier risk thinking*...

Mistakes in code may cease to exist, because the code containing them is rewritten, or the hardware on which the software runs is replaced. Correcting a coding mistake that is unlikely to result in a customer fault experienced is a wasted investment.

A study by Di Penta, Cerulo and Aversano<sup>400</sup> recorded the output from running various static analysis tools (Rats, Splint and Pixy) on each release of several large software systems (i.e., Samba, Squid and Horde). Comparing the tool output from each software release, it was possible to find the first/last release where a warning occurred for a particular line of code (or may still be occurring in the latest release).

Figure 6.5 shows the survival curve for the two most common warnings reported by Splint (memory problem and type mismatch, over 85% of all generated warnings). In the case of Splint type mismatches are more likely to be removed before memory problems first while for Squid memory problems are much more likely to be removed before type mismatches; also see Figure 11.76.

The average lifetime of coding mistakes varies between programs and kind of mistake.<sup>1206</sup>

Some hardware devices have a relatively short lifetime, e.g., mobile phones and graphics cards. Comparing the survival rate of reported faults in Linux device drivers and other faults in Linux shows that for the first 18 months, or so, the expected lifetime of reported faults in device drivers is much shorter than faults in other systems (see Figure 6.6); thereafter, the two faults lifetimes are roughly the same...<sup>1207</sup>

Also, see:<sup>?</sup>

emailed for fig 2 data...<sup>1549</sup>?

Files ported from forked versions of BSD contain fewer mistakes...<sup>1311</sup> emailed...

Number of devices submitted to the FDA, for approval, every year...?

Customer complaints...? Warrant claims on software sales... /home/data.../warranty-claims

While coding mistakes are exploited by computer viruses, causing business disruption, the greatest percentage of computer related loses come from financial fraud by insiders and traditional sources of loss such as theft of laptops and mobiles.<sup>1329</sup>

All mistakes have the potential to have costly consequences, but in practice most appear to be an annoyance. One study<sup>26</sup> found that only 2.6% of the vulnerabilities listed in the NVD have been used, or rather their use has been detected, in viruses and network threat attacks on computers.

emailed for data...?

?

A market has developed for faults that allow third parties to gain control of other peoples' computers<sup>25</sup> (e.g., spying agencies and spammers), and some vendors have responded by creating vulnerability reward programs. The list of published rewards are a measure of the value vendors place on this particular kind of fault experience. plot values<sup>498</sup> and?<sup>...</sup>

May be more cost effective to pay bug bounties than investing in finding mistakes prior to release...<sup>498</sup>...

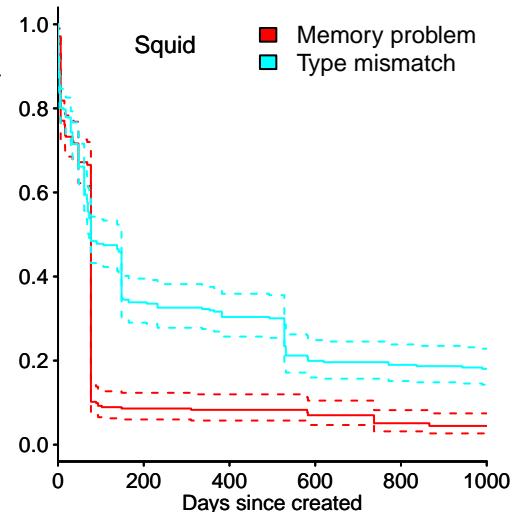
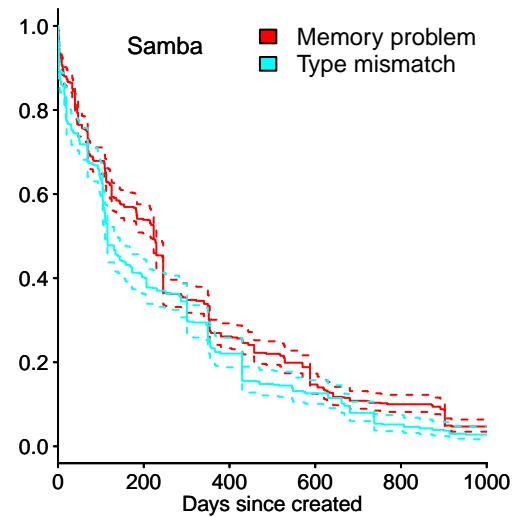


Figure 6.5: Survival curve of the two most common warnings reported by Splint in Samba and Squid. Data from De Penta et al.<sup>400</sup> code

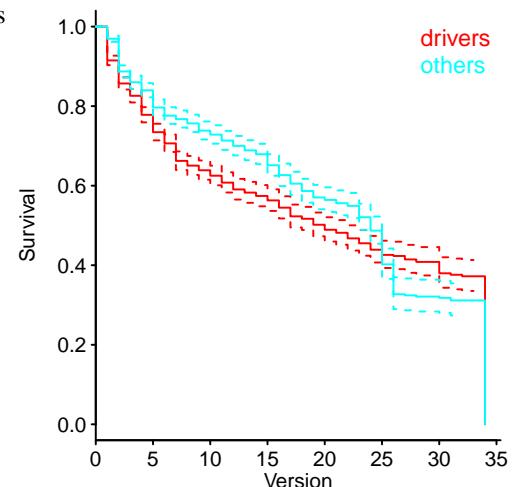


Figure 6.6: Survival rate of reported faults in Linux device drivers and other Linux subsystems... Data from Palix et al.<sup>1207</sup> code

Over half of those registered with a bounty program report a single bug, a quarter report two,<sup>1008</sup> also see Figure 4.39...

The *willingness-to-pay* approach attempts to determine the maximum amount that those at risk would individually be willing to pay for improvements to their or other people's safety. Each individual may only be willing to pay a small amount, but if they are a member of a group the amounts can be added together to estimate a value for the group "worth" of a safety improvement. For instance, assuming that in a group of 1.5 million people, each person is willing to pay £1 for safety improvements that achieve a 1 in 1 million reduction in the probability of death; the summed WTP *Value of Preventing a Statistical Fatality* (VPF) is £1.5 million (the same approach can be used to calculate *Value of Preventing non-fatal Injuries*).

If a customer reports a fault, in software they have purchased, what incentive does the vendor have to correct the problem and provide an update (they have the customers' money; assuming the software is not so fault ridden that it is returned for a refund)? Possible reasons include:

- it would be more expensive not to fix the coding mistake, e.g., the change in behavior caused by the fault experience could cause an accident, or negative publicity, that has an economic impact greater than the cost of fixing the mistake. Not wanting to lose money because a mistake had consequences that resulted in legal action...
- a customer support agreement requires certain kinds of reported faults to be fixed...
- public perception and wanting to maintain customer good will, in the hope of making further sales.

The time taken to fix publicly disclosed vulnerabilities is shorter than for vulnerabilities disclosed privately to the vendor; see the analysis in Section 11.11.3.1,

- it is fill-in work for developers between product releases...

Which implementation mistakes are corrected? While there is no benefit in correcting mistakes that customers are unlikely to experience, it may not be possible to deduce whether it is likely to produce a customer fault experience and so it has to be added to the list of known problems. Once a product has been released and known to be acceptable to many customers there may not be any incentive to actively search for potential fault experiences, consequently the only mistakes likely to be corrected are the ones reported by customers.

The vendor decision on whether to handle a reported fault is based on a cost/benefit trade-off involving the following factors:

- does a simple workaround exist, so users can continue using the software without needing an update, i.e., is it possible to reduce the issue to a low priority,
- the likely cost of correcting the mistake,
- the probability that a customer will experience a fault and recognize it as such,
- the impact to the vendor if a known mistake is not corrected. Adverse publicity when cost/benefit analysis made by a vendor is seen as callous. Example of Ford? being fined for not fixing a known problem because the cost/benefit was not great enough...

In some cases applications are dependent on the libraries supplied by the vendor of the host platform. One study<sup>969</sup> of Apps running under Android found that those Apps using libraries that contained more reported faults had a slightly smaller average user rating in the Google Play Store.

What motivates developers to fix reported faults in Open Source projects?

- they work for a company who provides software support services for a fee. Having a reputation as the go-to company for a certain bundle of packages is a marketing technique for attracting the attention of anybody looking to spend money on support services, custom modifications to a package or training. Correcting reported faults is a costly signal showing that somebody who knows what they are doing, i.e., status advertising,

- the desire to gain more users may be an incentive for the authors of a system, as a means of enhancing their reputation...
- developers dislike the thought of being wrong or making a mistake; a reported fault preys on their mind and may be fixed to make them feel better, also not responding to known problems in code is not socially acceptable behavior in software development circles. These feelings about what constitutes appropriate behavior are often enough to make developers want to spend time fixing mistakes in code they have written or feel responsible for, provided they have the time. I suspect a lot of problems get fixed by developers when their manager/wife thinks they are working on something more *useful*...

The developer's company is impacted by a fault Google vs. Apple... emailed for SQL...?

The cost of fixing a mistake, found prior to release, depends on where it is found in the development cycle... emailed for data...?

The higher the cost of system testing, the fewer opportunities there are likely to be to check systems at this level. Figure 6.7 shows the relationship between the unit cost of a missile (being developed for the US military), and the number of development test flights made.

## 6.3 Experiencing a fault

Software producing unintended behavior is a consequence of an interaction between a mistake in the code and particular input values.

A program's source code may be riddled with mistakes, but if typical user behavior does not cause the statements containing these mistakes to be executed, the program may gain a reputation for reliability. Similarly, there may only be a few mistakes in the source code, but if they are frequently experienced the program may gain a reputation for being fault-ridden.

Almost all existing research on software reliability has focused on the existence of the mistakes contained in source code. This is convenience sampling, large amounts of Open source is readily available, while information on the characteristics of program input is much harder to obtain.

The greater the number of people using a software system, the greater the volume and variety of inputs it has to process, consequently there are likely to be more faults reported.

A study by Shatnawi<sup>1416</sup> investigated the impact of the number of sites using a release of telecommunication switch software, on the number of software failures reported. A regression model fitted to the data shows that reported faults decreased over time and increase with the number of installed sites (see [reliability/2014-04-13.R](#)).

A study by Lucente<sup>982</sup> investigated help desk incident reports, from 800 applications used by a 100,000 employee company with over 120,000 desktop machines. Figure 6.8 shows...

A comparison of the number of faults reported in different software systems<sup>1251</sup> might be used to estimate the number of people using the different systems; any estimate of system reliability has to take into account the volume of usage and the likely distribution of input values.

Experiencing faults in widely used software is not difficult, e.g., the POSIX library on 15 operating systems<sup>397</sup>...

### 6.3.1 Input profile

The environments in which we live, and software systems operate, often experience regular cycles of activity; events are repeated with small variations, numeric values tend to have values about some mean... common occurrence of the Normal distribution in social science experiments...

Mistakes in code that interact with commonly occurring input values are likely to be noticed and fixed during development; beta testing is a method for discovering common customer usage that developers have not been testing against... some events in everyday life have been found to have an exponential distribution... Anderson library books...

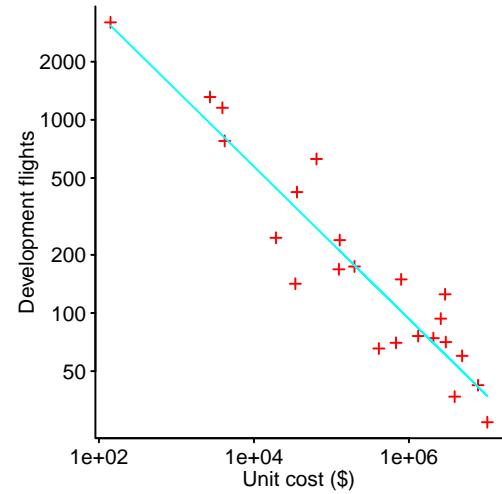


Figure 6.7: Unit cost of a missile against the number of development test flights it had. Data extracted from Augustine.<sup>68</sup> [code](#)

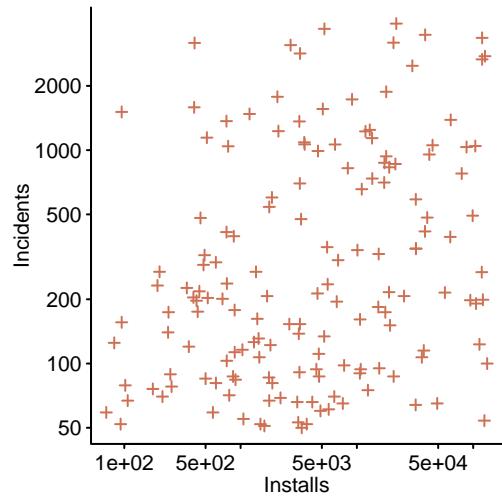


Figure 6.8: Number of reported incidents for each of 800 applications installed on over 120,000 desktop machines. Data from Lucente.<sup>982</sup> [code](#)

The input profile that resulted in the faults being experienced is an essential aspect of any analysis of program fault characteristics.

Unknown mistakes<sup>iii</sup> exist in shipped systems because the values needed to cause them to experience a fault are not included in the testing process.

Address traces illustrate how the execution characteristics of a program can be dramatically changed by its input. Figure 6.9 shows the number of memory accesses while executing gzip on two different input files. The small colored boxes representing 100,000 executed instruction on the x-axis and successive 4,096 bytes of stack on the y-axis, the colors denote number of accesses within the given block on a logarithmic scale.

A software system processes at least two distinct input profiles, the one used by developers to test it and the one driven by actual usage. Ideally the test and actual usage input profiles are the same, otherwise time and money is wasted fixing mistakes that the customer will not experience. However, even if it is possible to obtain accurate information on the customer input profile, it may be more cost effective to estimate it during testing.

The test process may include automatically generated input cases. Automated techniques include modifying existing tests and various forms of template guided random generation; terms used to denote these processes include *fuzzing* and *mutation*. Fuzzing is often used to test for a particular kind of fault experience, abnormal termination; some tools use a fuzzing selection strategy intended to maximise the likelihood of generating a file that causes a crash fault, e.g., CERT's BFF uses a search strategy that gives greater weight to files which have previously produced faults<sup>726</sup> (i.e., it is a biased random process).

Does the interaction between mistakes in the source code and an input profile result in any recurring patterns in the timing of fault experiences?

One way of answering this question is to count the number of inputs successfully processed by a program between successive fault experiences.

A study by Nagel and Skrivan<sup>1131</sup> investigated the timing characteristics of fault experiences in three programs, each written independently by two developers. During execution, each program processed inputs selected from the set of permissible values, when a fault was experienced its identity, execution time up to that point and number of input cases processed were recorded; the coding mistake was corrected and program execution continued until the next fault experience, until five or six faults had been experienced, or the mistake was extremely time-consuming to correct (the maximum number of input cases on any run was 32,808). This cycle was repeated 50 times, always starting with the original, uncorrected, program; the term *repetitive run modeling* was used to denote this form of testing.

Figure 6.10 shows the order in which distinct faults were experienced, by implementation A2 over 50 replications; edge values show the number of times the  $n^{th}$  fault was followed by a particular fault. For example, starting in state A2-0 fault 1 was the first fault discovered during 37 runs and this was followed by fault 3 in one run.

The number of input cases processed before a given number of faults was experienced, during the 50 runs of implementation A2 is shown in the upper plot of Figure 6.11; the lower plot shows the number of inputs processed before each of five distinct faults was experienced.

What is the likely number of inputs that have to be processed by implementation A2 for the sixth distinct fault to be experienced? A regression model could be fitted to the data seen in the upper plot of Figure 6.11, but a model fitted to a sample of five distinct fault experiences will have a wide confidence interval. There is no reason to expect that the sixth fault will be experienced after processing any number of inputs, there appears to be a change point after the fourth fault, but this may be random noise that magnified by the small sample size...

A study by Nagel, Scholz and Skrivan<sup>1130</sup> partially replicated and extended the previous study; there were three new developers, two implemented problem three from the first study (in assembler, rather than Fortran), and two implemented problem one from the first study and a new problem...

A study by Dunham and Pierce<sup>419</sup> replicated and extended the work of Nagel and Skriva; problem 1 was independently reimplemented by three developers. The three implementations were each tested with 500,000 input cases, when a fault was experienced the number of inputs processed was recorded, the coding mistake corrected and program execution restarted. This

<sup>iii</sup> Management may consider it cost effective to ship a system containing known mistakes.

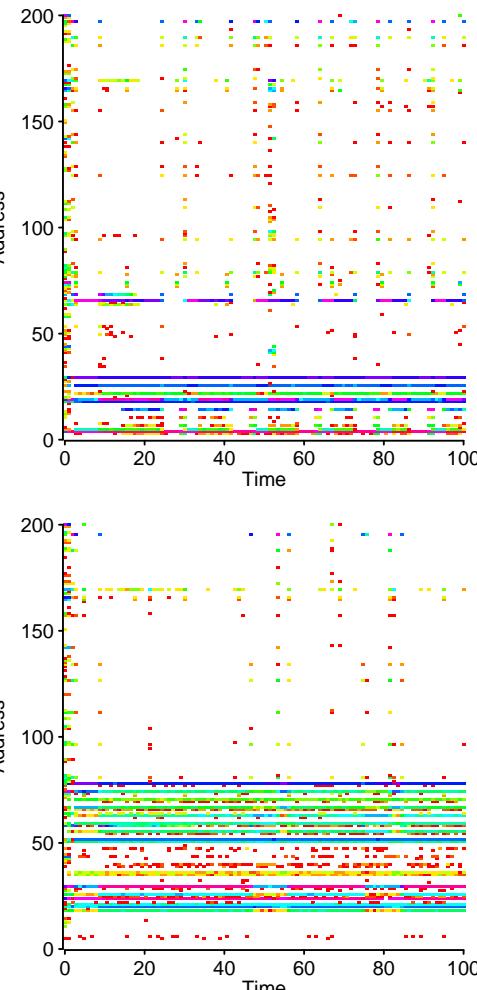


Figure 6.9: Number of accesses to memory address blocks, per 100,000 instructions, executing gzip on two different inputs. Data from Brigham Young<sup>210</sup> via Feitelson. [code](#)

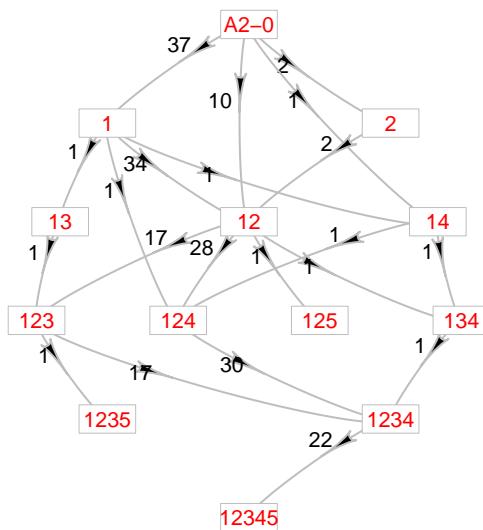


Figure 6.10: Transition counts of five distinct faults experienced in 50 runs of program A2; boxes labeled with the faults experienced up to that point. Data from Nagel et al.<sup>1131</sup> [code](#)

cycle was repeated four times, always starting with the original implementation, fixing and recording as faults were experienced.

Figure 6.12 shows the number of input cases processed, by two of the implementations (only one fault was ever experienced during the execution of the third implementation) before a given number of fault experiences, during each of the four runs. The grey lines are an exponential equation fitted using regression; these two lines show that as the number of faults experienced grows, more input cases are required to experience another fault, and that code written by different developers has different fault rates per input.

A second study by Dunham and Lauterbach<sup>418</sup> used 100 replications for each of the three programs, and found the same pattern of results seen in the first study.

Some published fault experience experiments have used time (computer or user), as a proxy for the quantity of input data. It is not always possible to measure the quantity of input processed, and time is often readily available.

A study by Wood<sup>1677</sup> analysed faults found by the product Q/A group in four releases of a subset of products. Figure 6.13 shows that the fault experience rate is similar for the first three releases (the collection of test effort data for release 4 is known to have been different from the previous releases)... .

A study by Pradel<sup>1276</sup> searched for thread safety violations, by 15 classes in the Java standard library and JFreeChart declared to be thread safe, and 8 classes in Joda-Time not declared to be thread safe; automatically generated test cases were used. Thread safety violations were found in 22 out of the 23 classes; for each case the testing process was run 10 times, and the elapsed time to discover the violation recorded (see Figure 6.14).

A study by Adams<sup>6</sup> investigated faults reported in applications over time. Figure 6.15 shows that approximately one third of all detected faults occurred on average every 5,000 years of execution time (over all users). Only around 2% of faults occurred every five years of execution time.

Multiple occurrences of the same fault provide information about the likelihood that the coding mistake will be executed and about the likelihood of encountering input that can trigger a fault. A regression model using a biexponential equation (e.g.,  $a \times e^{b \times x} + c \times e^{d \times x}$ , where  $x$  is the rank order of occurrences of each fault) has been fitted to the following crash fault data (also see Figure 11.52).

Why is a biexponential model such a good fit? A speculative idea is that the two exponentials are driven by the two independent processes involved in creating the data: the users writing source code, and the mistakes contained in the compiler. The users just happen to regularly write particular patterns of code and certain paths through the compiler are more likely than others; each of these two patterns of behavior drives a process that can be modeled by a single exponential.<sup>iv</sup>

A study by Zhao and Liu<sup>1705</sup> investigated the crash faults found by fuzzing the files processed by six Open source programs. Figure 6.16 shows the number of times unique crash faults were experienced in convert and autotrace (estimated by tracing back to a program location), along with lines fitted using biexponential regression models

The reliability of a system depends on the reliability of the components used to process the input. A rarely used component with poor reliability could have less impact on overall system reliability than a frequently used component with high reliability. See the analysis of gcc reliability for an example of using Markov chains to build a model of the reliability of a system based on its components... .

Error propagation matrix... ?

The consequences of fault experiences not being independent...<sup>237</sup> Error rate varies between programs performing the same function... Correlated failures...

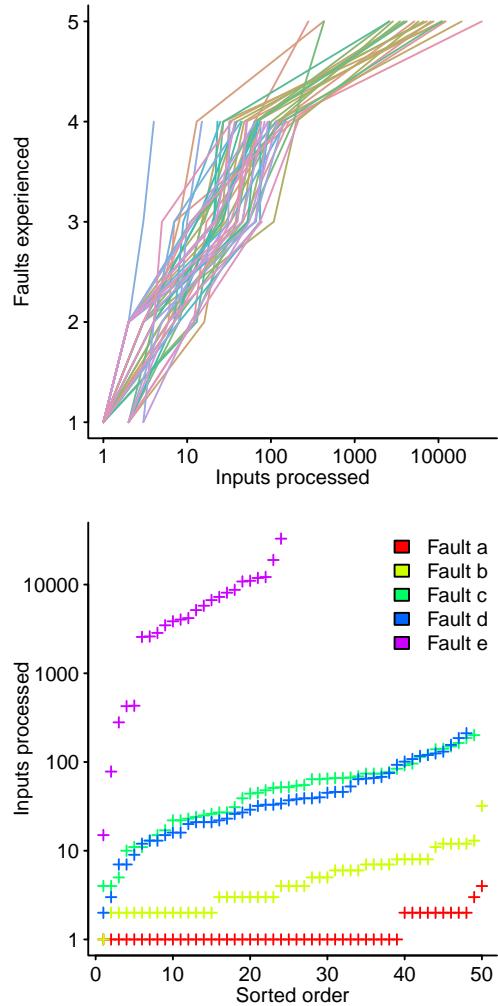


Figure 6.11: Number of input cases processed before a particular fault was experienced by program A2; the list is sorted for each distinct fault. Data from Nagel et al.<sup>1131</sup> code

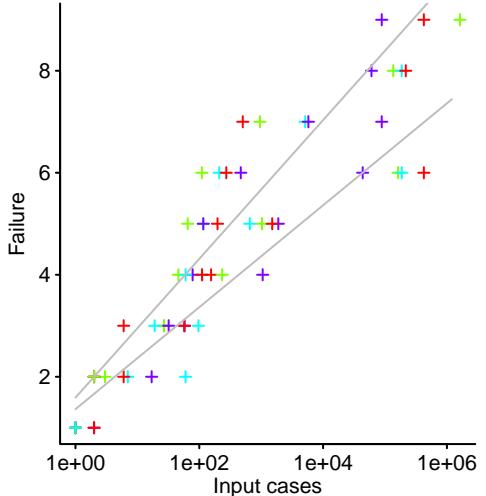


Figure 6.12: Input cases processed by two implementations, during four replications, before a failure occurred; grey lines are a regression fit for each program. Data from Dunham et al.<sup>419</sup> code

### 6.3.2 Further fault experiences — closed population

This section covers closed populations, i.e., no mistakes are added or removed; it also assumes that the characteristics of the input distribution remain constant.

<sup>iv</sup> Working out which process corresponds to which exponential appearing in the plots is left as an exercise to the reader (because your author has no idea).

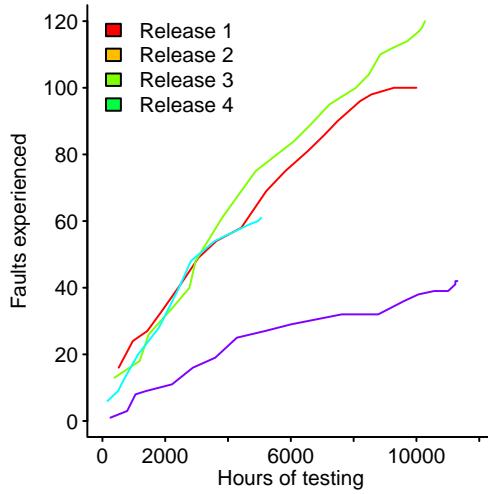


Figure 6.13: Fault discovered against hours of testing, for four releases of a product. Data from Wood.<sup>167</sup> [code](#)

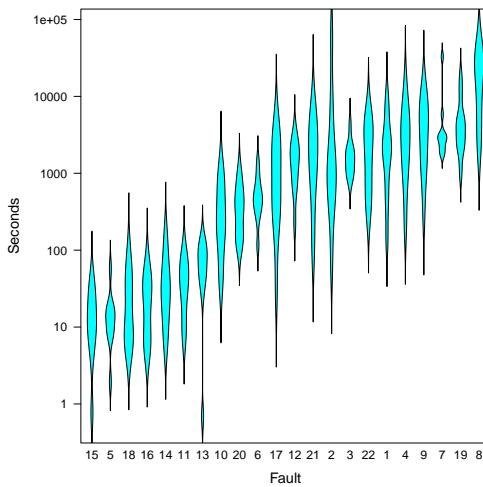


Figure 6.14: Time taken to discover a thread safety violation in 22 Java classes, violin plots for 10 runs involving each class. Data kindly supplied by Pradel.<sup>1276</sup> [code](#)

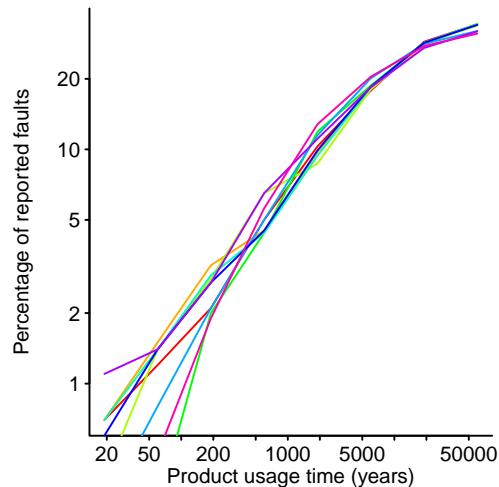


Figure 6.15: Percentage of reported problems having a given mean time to first problem occurrence (in months, summed over all installations of a product) for none products. Data from Adams.<sup>6</sup> [code](#)

After  $N$  distinct faults have been experienced, what is the probability that there exists new, previously unexperienced, faults?

Data on reported faults commonly takes two forms: incidence data (e.g., a record of the date of first report and no information on subsequent reports involving the same fault experience), and abundance data (e.g., a record of every fault experience).

It is not possible to use incidence data to distinguish between different exponential order fault growth models<sup>1082</sup> (which is nearly all models that have been proposed over the years as potentially applying to a software system). Modeling using incidence data requires samples from multiple sites (e.g., faults experienced within different companies, who use the software, tagged with location-id for each fault experience). It is often possible to fit a variety of equations to fault report data, using regression modeling; however, predictions about future fault experiences made using these models is likely to be very unreliable (see Figure 11.49).

When abundance data is available, the models discussed in Section 4.3.2 can be used to estimate the number of unique items within a population, and the number of new unique items likely to be encountered with additional sampling.

A study by Kaminsky, Eddington and Cecchetti<sup>815</sup> investigated crash faults in three releases of Office and OpenOffice (plus other common document processors), using fuzzing. Figure 6.17 shows a prediction of the growth in crash faults found in the 2003, 2007 and 2010 releases of Microsoft Office, along with 95% confidence intervals. Later versions are estimated to contain fewer crash faults, although the confidence interval for the 2010 release becomes rather wide.

Figure 6.18 shows the number of duplicate crashes experienced when the same fuzzed files were processed by the 2003, 2007 and 2010 releases of Microsoft Office. The blue/purple lines are the two parts of fitted biexponential models...

?

The previous analysis is based on information about faults that have been experienced. What is the likelihood of a fault experience, given that no faults have been experienced in the immediately previous time,  $T$ ?

An analysis by Bishop and Bloomfield<sup>166</sup> derives a lower bound for the reliability function,  $R$ , for a program executing without failure for time  $t$  after it has executed for time  $T$  without failure; it is assumed that the input profile does not change during time  $T + t$ . The reliability function is:

$$R(t|T) \geq 1 - \frac{t}{T+t} e^{-\frac{T}{T+t} \log(1+\frac{t}{T})}$$

If  $t$  is much smaller than  $T$ , this equation can be simplified to:

$$R(t|T) \geq 1 - \frac{t}{(T+t) \times e}$$

For instance, if a program is required to execute for 10 hours with reliability 0.9999, the initial failure free period, in hours, is:

$$\begin{aligned} 0.9999 &\geq 1 - \frac{10}{(T+10) \times e} \\ T &\geq \frac{10}{(1-0.9999) \times e} - 10 \approx 36,778 \end{aligned}$$

If  $T$  is much smaller than  $t$ , the general solution can be simplified to:

$$R(t|T) \geq \frac{T}{t}$$

How can this worst case analysis be improved on?

If it is assumed that a system executes  $N$  times without a failure, and has a fixed probability of failing,  $p$ , the probability of one or more failures occurring in  $N$  executions is (and then assuming  $N$  is large):

$$C = \sum_{n=1}^N p(1-p)^{n-1} \approx p \frac{1-(1-p)^N}{1-(1-p)}$$

How many executions, without failure, need to occur to have a given confidence that the actual failure rate is below a specified level? Rearranging, we get:

$$N = \left\lceil \frac{\log(1 - C)}{\log(1 - p)} \right\rceil$$

Plugging in values for confidence,  $C = 0.99$ , and failure probability,  $p < 10^{-4}$ , then the system has to execute without failure for 46,050 consecutive runs.

This analysis is not realistic because it assumes that the probability of failure,  $p$ , remains constant for all input cases; studies show that  $p$  can vary by several orders of magnitude.

Multiple reports of the same vulnerability... <sup>688</sup>

### 6.3.3 Further fault experiences — open population

In an evolving system, existing coding mistakes are corrected and new ones are made; new features may be added that interact with existing functionality, i.e., there may be a change of behavior for the same input. The population of mistakes is open, and the input distribution may be evolving.

Bug trackers for Open source projects often contain multiple reports for the same underlying coding mistake, but the population contained within bug tracking systems may be changing because the software system is changing, the user base is changing (i.e., the number of users running a particular version changes as people migrate to a newer release), or both.

Techniques for estimating the characteristics of open populations include: survival analysis of members, ...

How well do sums of exponentials fit the data?

A study by Sun, Le, Zhang and Su<sup>1518</sup> investigated the bugs reported in GCC and LLVM. Figure 6.19 shows the number of times distinct faults reported in GCC (from 1999 to 2015), with a fitted biexponential and the two component exponentials.

A study by Sadat, Bener and Miranskyy<sup>1368</sup> investigated duplicate bug reports in Apache, Eclipse and KDE over 18-years. Figure 6.20 shows the number of times distinct faults reported in KDE, with a fitted triexponential and the three component exponentials. The third component exponential...

Successive releases of a software system often include large amounts of code from earlier releases. The collection of source code that is new in each release can be treated as a distinct ecosystem containing a closed population of mistakes; these ecosystems do not grow but can shrink (when code is deleted). All the code contained in the first release is the foundation ecosystem.

The number of faults that could be experienced in a version of a software system is the sum of the estimated fault experiences that might be triggered by the mistakes in code it contains from the current and earlier releases.

A study by Ozment and Schechter<sup>1203</sup> investigated reported faults in 15 successive versions of OpenBSD, the first in May 1998. The version history of the source was extracted for each release and 140 vulnerabilities (from NVD and several other sources) traced back to the source/version containing the coding mistake.

Table 6.2 shows... not enough data points...

A study by Massacci, Neuhaus and Nguyen<sup>1026</sup> investigated 899 Security Advisories in Firefox, reported against six major releases. Their raw data is only available under an agreement that does not permit your author to directly distribute it to readers; the data used in the following analysis was reverse engineered from the paper or extracted by your author from other sources.

The following analysis attempts to build a model of the relationship between the age of code, end-user source code usage and reported faults.

Table 6.3 shows the lowest version (columns) of Firefox containing the mistake in the source code and the highest version (rows) to which a corresponding fault report applies. For instance, 42 faults were discovered in version 2.0 that relate to mistakes assumed to have been made in source code written for version 1.0. Only corrected coding mistakes have been counted, unfixed mistakes are not included in the analysis. Firefox versions have a release and retirement date after which the version is no longer supported (i.e., no more coding mistakes corrected in that version).

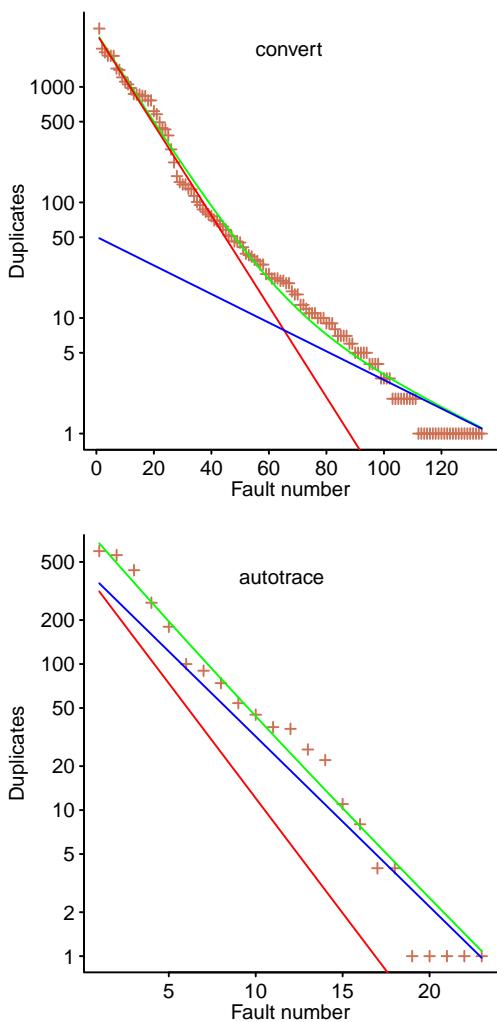


Figure 6.16: Number of times the same fault was experienced in two programs, crashes traced to the same program location; with fitted biexponential equation. Data kindly provided by Zhao.<sup>1705</sup> [code](#)

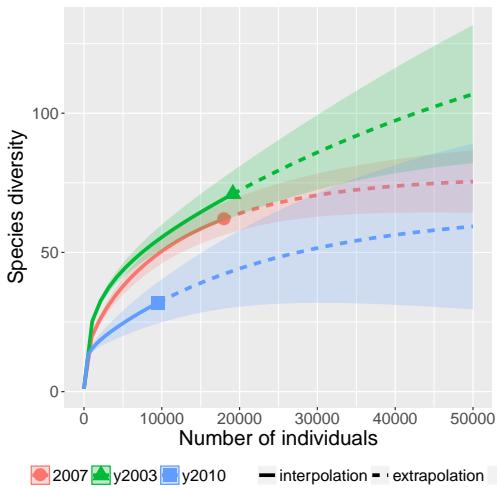


Figure 6.17: Predicted growth, with 95% confidence intervals, in the number of new crash faults found in the 2003, 2007 and 2010 releases of Microsoft Office. Data from Kaminsky et al.<sup>815</sup> [code](#)

	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	Total
2.3	5															5
2.4	11															11
2.5	6		1													7
2.6	5	1														6
2.7	12	4	2	2	2											22
2.8	12	1		1	2											16
2.9	4			2												6
3.0	3	1			1		2									7
3.1	8	2	1	2				1	1							15
3.2	6	2					1	2		1						12
3.3	2	1		2							2					7
3.4	2				1		1		1							5
3.5	7	1	1				2		1			1				13
3.6	3		1													4
3.7	1	1		1								1				4

Table 6.2: Reported vulnerabilities in a sequence of versions of OpenBSD (coding mistake made in column version, fault reported in row version). Data from Ozment et al.<sup>1203</sup>

	1.0	1.5	2.0	3.0	3.5	3.6
1.0	79					
1.5	71	108				
2.0	42	104	126			
3.0	97	15	22	67		
3.5	32			30	32	
3.6	13		1	5	41	

Table 6.3: Number of reported security advisories in versions of Firefox; coding mistake made in version columns, advisory reported in version row. Data from Massacci et al.<sup>1026</sup>

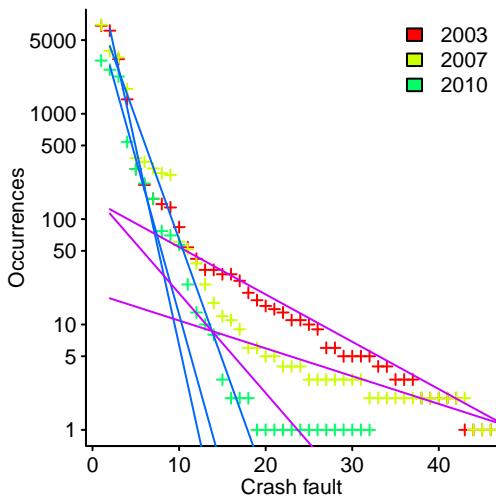


Figure 6.18: Number of program crashes traced to the same executable location, in the 2003, 2007 and 2010 releases of Microsoft Office (blue/purple lines are the two parts of biexponential fits). Data from Kaminsky et al.<sup>815</sup>

How many users does each version of Firefox have at any time?

Figure 6.22 shows the market share of the six versions of Firefox between official release and end-of-support. Estimated values appear to the left of the vertical grey line, values from measurements to the right; at its end-of-support date version 2.0 still had a significant market share.

Assuming everybody who uses the Internet uses a browser; Figure 6.23 shows the growth of internet usage over time, broken down by nation development status.

The end-user usage of source code originally written for a particular version of Firefox, over time, is calculated as follows:

- number of lines of code originally written for a particular version contained within the code used to build a later version, or that particular version; call this the build version,
- multiplied by the market share of the build version,
- multiplied by the number of Internet users (the developed world count is used).

Figure 6.24 is an example using the source code originally written for Firefox version 1.0. The green points are the code usage for version 1.0 code executing in Firefox build version 1.0, the orange points the code usage for version 1.0 code executing in build version 1.5 and so on to the yellow points, which is the code usage for version 1.0 code executing in build version 3.6. The black points are the sum over all build versions.

Much of the overall growth comes from growth in Internet usage, and in the early years there is also substantial growth in browser market share.

This analysis ignores the possibility that the browsing habits of people who started using the Internet in 2004 may be different from those who first started in 2010, such as time on spent on the Internet, the propensity to report a fault and cultural differences (e.g., European users vs. Chinese users). The contents of web pages also changed...

Combining information on the age of code, end-user source code usage and reported faults, we get...

## 6.4 Where is the mistake?

Information on where mistakes are made during the development of software systems can be used to focus resources on the major problem areas. A few studies<sup>347</sup> have measured across top-level entities such as project phase (e.g., requirements, coding, testing, documentation), while others have focused on specific components (e.g., source code, configuration file), or low level constructs (e.g., floating-point<sup>398</sup>).

The root cause of a mistake, by a person, may be knowledge based (e.g., incorrect beliefs about the semantics of the programming language being used), or it may be skill based (e.g., an error was made in the implementation...)<sup>1316</sup>

Mistakes in hardware<sup>280, 410, 751</sup> tend to occur much less frequently than mistakes in software, and mistakes in hardware are not considered here<sup>v</sup>

The *user interface*, the interaction between people and an application, can be a source of fault experiences in the sense that a user misinterprets correct output, or selects a parameter option that causes program behavior that was not intended.

The mistake that causes a fault experience may be in the environment in which a program is built and executed. Many library package managers support the installation of new packages via a command line tool. One study<sup>1566</sup> made use of typos in the information given to command line package managers to cause a package other than the one intended to be installed.

emailed...?

emailed...??

### 6.4.1 Requirements

A requirements mistake is made when one or more requirements are incorrect, inconsistent or incomplete; an ambiguous specification<sup>154</sup> is a potential mistake. The number of mistakes contained in requirements may be of the same order of magnitude<sup>347</sup> or exceed the number of mistakes found in the code,<sup>1303</sup> different people bring different perspectives to requirements analysis, which can result in them discovering different problems.<sup>929</sup>

The same situation can be approached from multiple viewpoints, depending on the role of the viewer; see Figure 2.47. Those implementing a system may fail to fully appreciate all the requirements implied by the specification they are given; context is very important, see Figure 2.46.

Software systems are implemented by generating and interpreting language (human and programming). Reliability is affected by the people's variability in their use of language,<sup>157</sup> what they consider to be correct English syntax<sup>1480</sup> and the interpretation of numeric phrases. Language issues are covered in Section 6.4.2, on source code.

The impact of failing to correctly implement a requirement include: going unnoticed, being embarrassing and causing loss of life.

During the lifetime of a project, existing requirements are misinterpreted or changed, and new requirements are added.

Figure 8.28 shows that many mistakes are corrected by modifying the requirements. In those cases where various kinds of behavior are equally acceptable, modifying the requirements documents may be the cheapest path to resolving a mistake...

A study by van der Meulen, Bishop and Revilla<sup>1592</sup> investigated the coding mistakes in 29 thousand implementations of the  $3n + 1$  problem; the programs had been submitted to a programming contest. All submitted implementations were tested and programs producing identical outputs were assigned to the same equivalence class (competitors could make multiple submissions, if the first failed to pass all the tests). In many cases the incorrect output, for an equivalence class, could be explained by a failure of the competitor to implement a requirement implied by the problem being solved, e.g., failing to swap input number pairs when the first was larger than the second.

<sup>v</sup> Your author once worked on a compiler for a cpu that was still in alpha release; the generated code was processed by a sed script to handle known problems in the implementation of the instruction set, problems which changed over the weeks as updated versions of the cpu became available.

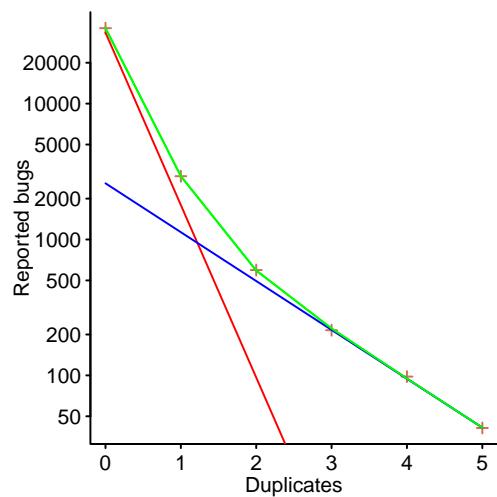


Figure 6.19: Number of instances of the same reported fault in GCC, with fitted biexponential regression model. Data from Sun et al.<sup>1518</sup> [code](#)

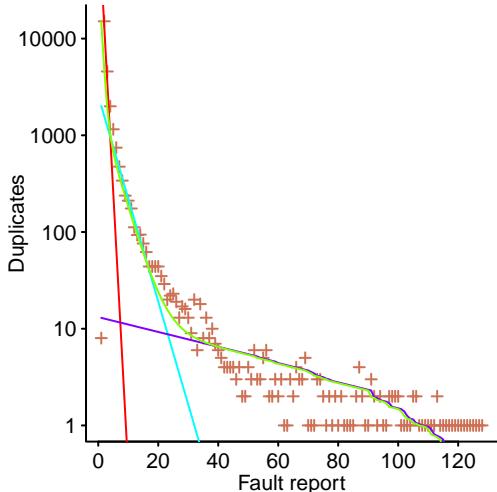


Figure 6.20: Number of instances of the same reported fault in KDE, with fitted triexponential regression model. Data from Sadat et al.<sup>1368</sup> [code](#)

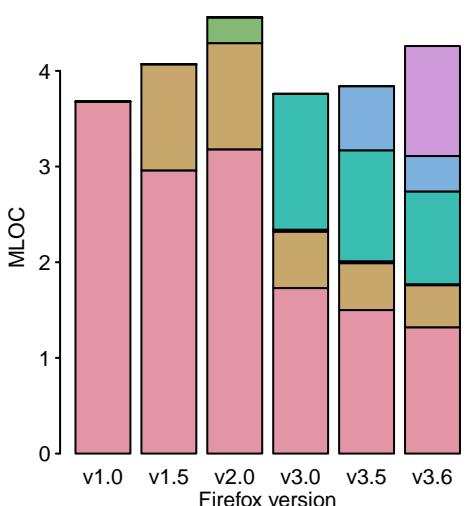


Figure 6.21: Amount of source (millions of lines) in each version broken down by the version in which it first appears. Data extracted from Massacci et al.<sup>1026</sup> [code](#)

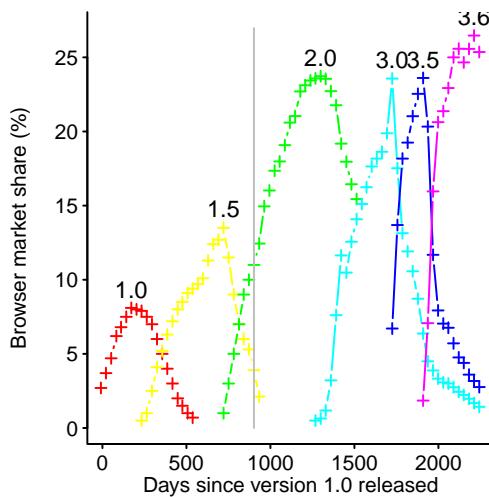


Figure 6.22: Market share of Firefox versions between official release and end-of-support. Data from Jones.<sup>7</sup> [code](#)

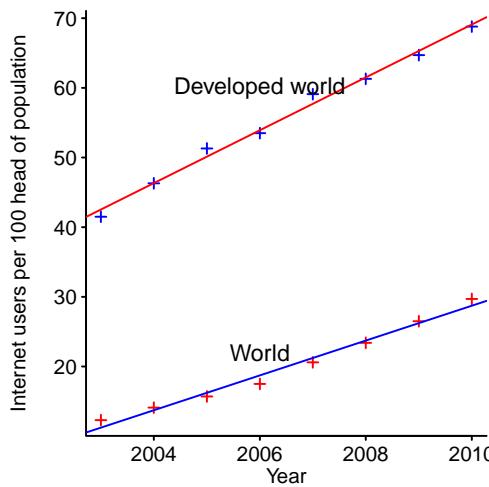


Figure 6.23: Number of people with Internet access per 100 head of population in the developed world and the whole world. Data from ITU.<sup>752</sup> [code](#)

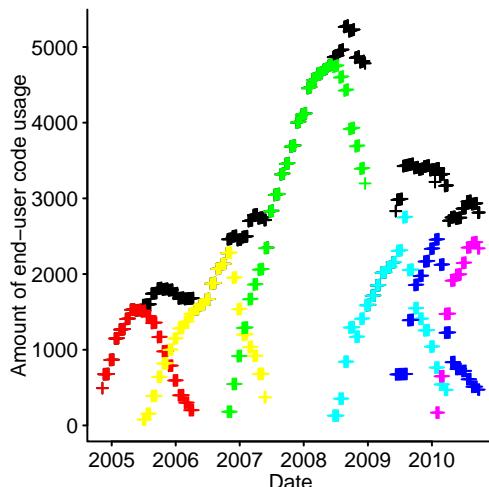


Figure 6.24: Amount of end-user usage of code originally written for Firefox version 1.0 by various other versions. Data extracted from Massacci et al.<sup>1026</sup> [code](#)

Figure 6.25 shows the 36 equivalence classes containing the most members; the most common is the correct output, followed by always returning 0 (zero).

The publicly available studies of requirements errors include a study of the software for the Voyager and Galileo spacecraft<sup>993</sup> and ...

ISO processes require that the committee responsible for a standard maintain a log of reported defects and the committee's response. Figure 6.26 shows the growth in various kinds of defects reported against the POSIX standard (ISO 9945). ISO language DRs over time...

There have been very few studies<sup>1427</sup> of the impact of the form of specification on its implementation.

Change impact analysis during maintenance...<sup>367</sup>

A connection should exist between lines of code and requirements in that a line of codes exists because of one or more requirements in the specification (ignoring mistakes that result in dead or unreachable code)...<sup>1676</sup>...

## 6.4.2 Source code

Source code is the focus of much research: it is the original form of an executable program that experiences faults, and is usually what is modified by developers to correct mistakes. From the research perspective it is now available in bulk, and techniques for analysing it are known, and practical to implement.

There are recurring patterns in the changes made to source code to correct mistakes,<sup>1211</sup> one reason for this is that some language constructs are used more often than others.<sup>785</sup> The idea that there is an association between reported faults and particular usage patterns in source code, or program behavior, is popular; over 40 association measures have been proposed.<sup>984</sup>

Errors of omission can cause faults to be experienced. One study<sup>640</sup> of error handling by Linux file systems found that 13% of calls ignore an error code, i.e., do not handle it. Cut-and-paste is a code editing technique that is susceptible to errors of omission, that is, failing to make all the necessary modifications to the pasted version.<sup>137</sup> Significant numbers of cut-and-paste errors have been found in JavaDoc documentation<sup>1199</sup>...

Patterns of source code usage are also seen in the kinds of mistakes made. Figure 6.27 shows the ranked occurrences of each kind of compiler message generated by Java and Python programs submitted by students.

?

Proponents of particular languages sometimes claim that programs written in the language are more reliable (other desirable characteristics may also be claimed), than if written in other languages. Most experimental studies comparing the reliability of programs written in different languages have either used students<sup>580</sup> or had little statistical power... A language may lack a feature that, if used, would help to reduce the number of mistakes made by developers, e.g., support for function prototypes in C,<sup>1408</sup> which were added to the language in the first ANSI Standard.

Are programs written in some languages more susceptible to experiencing a fault than others? A study by Spinellis, Karakoidas and Lourida<sup>1478</sup> made various kinds of small random changes to 14 different small programs, each implemented in 10 different languages (400 random changes per program/language pair). The ability of these modified programs to compile, execute and produce the same output as the unmodified program was recorded.

Figure 6.28 shows the fraction of programs that compiled, executed and produced correct output, for the various languages. There appear to be two distinct language groupings, each having similar successful compilation rates; one commonality of languages in each group is requiring, or not, variables to be declared before use. The data from the study does not contain enough information to investigate this, or other, possibilities.

The fitted regression model (see [reliability/fuzzer/fuzzer-mod.R](#)) contains an interaction between language and program (the problems implemented did not require many lines of code and in some cases could be solved in a single line in some languages), and a logarithmic dependency on program length (i.e., number of lines).

Another study<sup>115</sup> modified one statement of nine medium-sized Java programs (using abstract syntax trees)...

A study by Aman, Amasaki, Yokogawa and Kawahara<sup>34</sup> investigated time to bug-fix events, in the source files of 50 projects implemented in Java and 50 in C++. The survival time of files (i.e., time to reported fault causing the source to be modified) was the same for both languages, and number of developers, and number of project files had almost zero impact on a Cox model; see [survival/Profes2017-aman.R](#).

Various metrics have been proposed as measures of some desirable, or undesirable, characteristic of a unit of code, e.g., a function. Halstead's and McCabe's cyclomatic complexity are perhaps the most well-known such metrics (see Section 7.2.8), both count the source contained within a single function. Irrespective of whether either of these metrics correlate with anything other than what they directly represent,<sup>911</sup> they can be easily manipulated by splitting functions with high values into two or more functions, which then have lower values of the metric (just as it is possible to reduce the number of lines of code in a function, by putting all the code on one line).

The value of McCabe's complexity (number of decisions, plus one) for the following function is 5, and there are 16 possible paths through the function:

```
int main(void)
{
if (W) a(); else b();
if (X) c(); else d();
if (Y) e(); else f();
if (Z) g(); else h();
}
```

each if...else contains two paths and there are four in series, giving  $2 \times 2 \times 2 \times 2$  paths. Restructuring the code, as below, removes the multiplication of paths caused by the sequence of if...else:

```
void a_b(void)
    {if (W) a(); else b();}
void c_d(void)
    {if (X) c(); else d();}
void e_f(void)
    {if (Y) e(); else f();}
void g_h(void)
    {if (Z) g(); else h();}

int main(void)
{
a_b();
c_d();
e_f();
g_h();
}
```

reducing mains McCabe complexity to 1, with the four new functions each having a McCabe complexity of two. Where has the complexity that main used to have gone? It now 'exists' in the relationship between the functions, a relationship that is not included in the McCabe complexity calculation. The number of paths that can be traversed, by a call to main, has not changed.

A metric that specifies a value for single functions (i.e., its value is calculated from the contents of individual functions) cannot be specified as a control mechanism (i.e., require that values not exceed some limit) because its value can be easily manipulated by moving contents into newly created functions. The software equivalent of what is known as *accounting fraud* in accounting.

Predictions are sometimes attempted<sup>1431,1715</sup> at the file level of granularity, e.g., predicting which files are more likely to contain mistakes that become fault experiences; the idea being that the contents of some files be rewritten. Any reimplementation will include mistakes and the cost of rewriting the code may be larger than fixing reported faults in the original code as they are discovered.

The idea that there is an optimal value for the number of lines of code in a function body has been an enduring meme (when object oriented programming became popular, the meme mutated to cover optimal class size). See Figure 8.38 for a discussion of the U-shaped defect

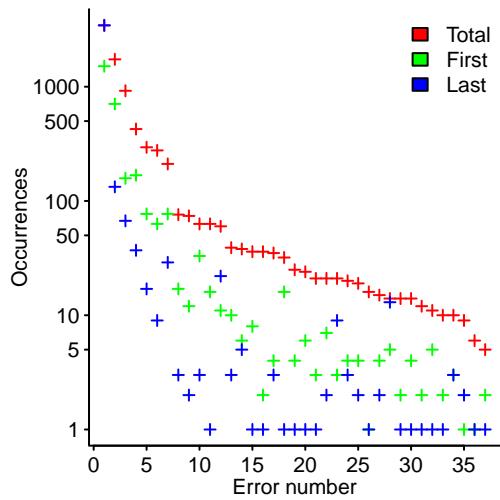


Figure 6.25: Total number of implementations in each of 36 equivalence classes, plus both first and last competitor submissions. Data from van der Meulen et al.<sup>1592</sup> [code](#)

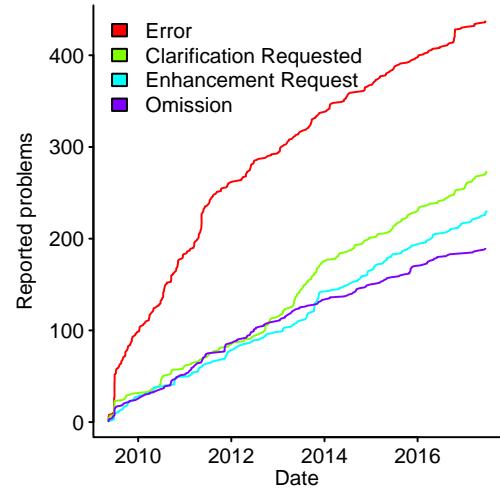


Figure 6.26: Cumulative number of defects logged against the POSIX standard, by defect classification. Data kindly provided by Josey.<sup>1189</sup> [code](#)

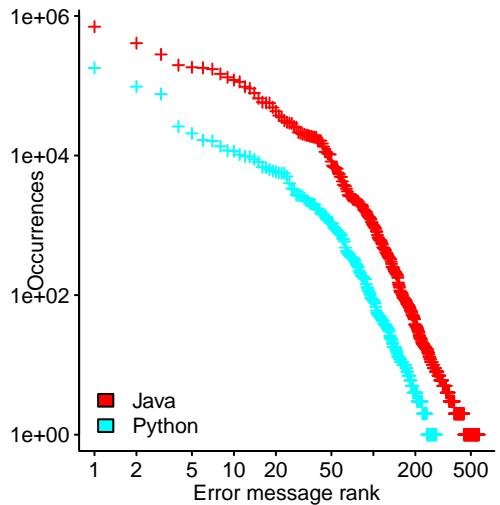


Figure 6.27: Ranked occurrences of compiler messages generated by submitted student Java and Python programs. Data from Pritchard.<sup>1284</sup> [code](#)

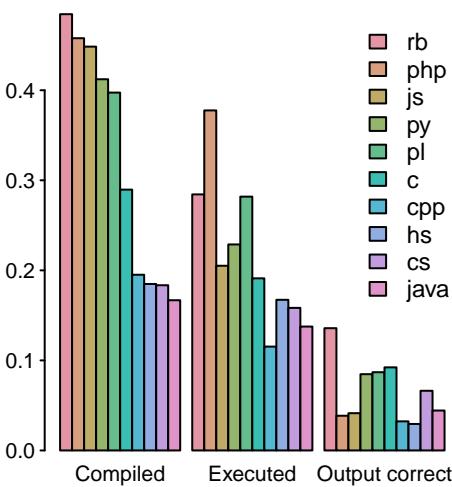


Figure 6.28: Fraction of mutated programs, in various languages, that successfully compiled/executed-produced same output. Data from Spinellis et al.<sup>1478</sup> code

density paper chase; other studies<sup>879</sup> investigating the relationship between reported faults and number of lines of code have failed to include usage information (i.e., number of people using the software) in the model.

Fixing most reported faults involves changing a few lines in a single function or changes within a single file. A study<sup>599</sup> of over 1,000 projects for each of C, Java, Python and Haskell found that correcting most coding mistakes involved adding and deleting a few tokens.

?

A study by Lucia<sup>983</sup> investigated localization techniques. Figure 6.29 shows the percentage of reported faults whose correction involved a given number of files, modules or lines; lines are power laws fitted using regression.

A study by Zhong and Su<sup>1709</sup> investigated commits associated with reported faults in five large Open source projects. Figure 6.30 shows the number of files modified while fixing reported faults, against normalized number of commits made while making these changes; grey line is the equation:  $100 \times \text{files}^{-2.1}$  (which is a reasonable fit to four of the five systems).

- Files
- Modules
- Lines

Faults (percent)

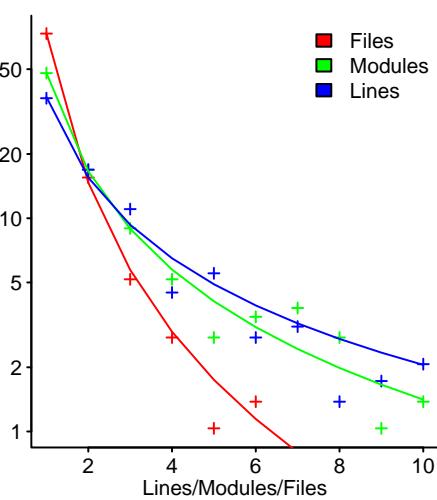


Figure 6.29: Number of reported faults whose fixes involved a given number of files, modules or lines in a sample of 290 faults in AspectJ; lines are power laws fitted using regression. Data from Lucia.<sup>983</sup> code

Software that is used evolves over time (e.g., lines are added/modified/deleted), and open source programs for which reasonable amounts of data are available are still evolving. See Figure ??... .

When existing code is changed, there is some probability that a mistake will be made.

A study by Purushothaman and Perry<sup>1288</sup> investigated small source code changes made to one subsystem of the 5ESS telephone switch software (4,550 files containing almost 2MLOC, with 31,884 modification requests after the first release changing 4,293 of these files). Figure 6.31 is based on an analysis of reported faults traced to updates, that involved modifying/inserting a given number of lines, and shows the percentage of each kind of update that resulted in a reported fault.

Some languages have a mechanism for developers to specify properties, about variables appearing in the source, that are expected to be true, e.g., `assert(x > y)...` ??

### 6.4.3 Build system

Using an inconsistent set of configuration options, when building an application, can generate programs that fail unpredictably. Variability models are used to express constraints between options.?<sup>9</sup> The Linux 2.6.33.3 kernel supports 6,918 configuration options, making possible over  $10^{23,563}$  option combinations; a random sample of 1,000,000 different option combinations failed to find any that were valid according to the variability model<sup>959</sup> (a random sampling of the more than  $10^{1,377}$  possible option combinations supported by OpenSSL found that 3% were considered valid).

Various techniques have been proposed for obtaining a sample of valid configuration options,<sup>814</sup> which might then be used for testing different builds of an application or analyzing the source code.<sup>1620</sup>

### 6.4.4 Third-party libraries

The availability of many open source libraries can make it more cost effective to interface to a third-party, rather than implementing a bespoke solution.

A study by Decan, Mens and Constantinou<sup>381</sup> investigated the time taken for security vulnerabilities in npm packages to be fixed, along with the time taken to update packages that depended on a version of a package having a reported vulnerability. Figure 6.32 shows survival curves (with 95% confidence bounds), for high and medium severity vulnerabilities, of time to fix a reported vulnerability (Base), and time to update a dependency (Depend) to a corrected version of a package.

?

### 6.4.5 Correctness vs. consistency

Intel trig functions...

Testing the quality of random number generators<sup>932</sup>

Errors in Microsoft Excel's random number generator<sup>1041</sup>

### 6.4.6 Documentation

Documentation is a cost paid today, that may provide a benefit later for the somebody else or the author.

If user documentation specifies functionality that is not supported by the software, the vendor may be liable to pay damages to customers expecting to be able perform the documented functionality.<sup>821</sup> Non-existent documentation is not unreliable, but documentation that has not been updated to match changes to the software is.

There have been relatively few studies of the reliability of documentation. A study by Rubio-Gonzalez and Llibit<sup>1361</sup> found 1,784 undocumented error return codes in an analysis of the source code of 53 Linux file systems. A study by Ma, Liu and Forin<sup>996</sup> tested an Intel x86 cpu emulator and found a wide variety of errors in the documentation specifying the behavior of the processor.

## 6.5 Non-software causes of unreliability

Hardware is built from moving parts and wears out. Electronic components operate by controlling the direction of movement of electrons; when the movement is primarily in one direction, atoms migrate in that direction, and over time this migration degrades device operating characteristics.<sup>1483</sup> A factor with ever smaller transistors is their decreasing mean time to failure; expected chip lifetimes have dropped from 10 years to 7, and decreasing.<sup>1655</sup>

As the size of components shrinks, and the number of components on a device increases, the probability that thermal noise will cause a bit to change state increases.<sup>854</sup>

Faulty hardware does not always noticeably change the behavior of a program when it is executing, apparently correct program execution can occur, e.g., image processing.<sup>1170</sup>

See Table 10.6 for a discussion of system failure traced to either cpu or DRAM failures and Section 10.2.3 for a study that looked for a correlation between checking for a hardware performance and likelihood of experiencing some intermittent faults.

A software reliability problem that was rarely encountered outside of science fiction a few decades ago, now regularly occurs in modern computers: cosmic rays (plus more local sources of radiation, such as the materials used to fabricate a device) flipping the value of one or more bits in memory or a running processor. Techniques for mitigating the effects of a radiation induced events have been proposed.<sup>1124</sup>

The two main sources of radiation are alpha-particles generated within the material used to fabricate and package devices and Neutrons generated by Cosmic-rays impacting with the upper atmosphere.<sup>73</sup> The data in Figure 6.33 comes from monitoring equipment located in the French Alps; either, 1,700m under the Fréjus mountain (i.e., all radiation is generated by the device), or on top of the Plateau de Bure at an altitude of 2,552m (i.e., radiation sources are local and Cosmic).<sup>72</sup> For confidentiality reasons the data has been scaled by a small constant.

Figure 6.33 shows how the number of bit-flips increased over time (measured in Mega-bits per hour), for SRAM fabricated using 130 nm, 65 nm and 40 nm processes. The 130 nm and 65 nm measurements were made underground, and the lower rate of bit-flips for the 65 nm process is the result of improved materials selection, reducing alpha-particle emissions; the 40 nm measurements were made on top of the Plateau de Bure and show the impact of external radiation sources.

The soft error rate is usually quoted in FITs (Failure in Time), with 1 FIT corresponding to 1 error per  $10^9$  hours per megabit, or  $10^{-15}$  errors per bit-hour. Consider a system with 4 GB of DRAM (1000 FIT/Mb is a reasonable approximation for commodity memory,<sup>1546</sup>

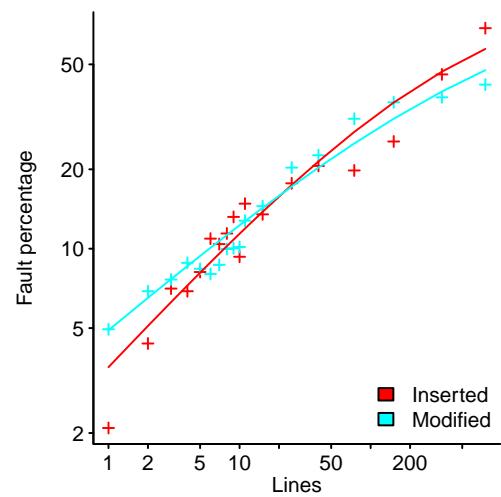


Figure 6.31: Percentage of insertions/modifications of a given number of lines resulting in a reported fault; lines are fitted regression models. Data from Purushothaman et al.<sup>1288</sup> code

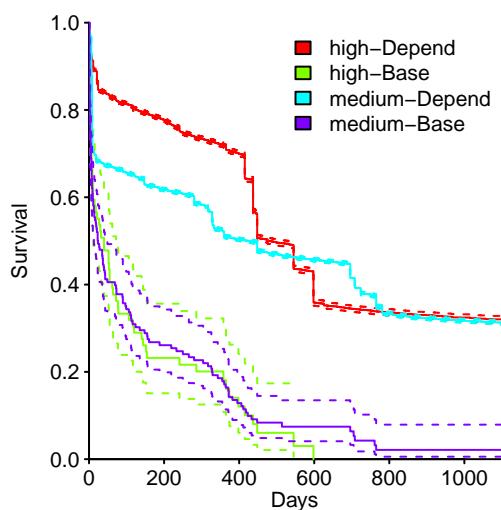


Figure 6.32: Survival curve (with 95% confidence bounds) of time to fix vulnerabilities reported in npm packages (Base) and time to update a package dependency (Depend) to a corrected version (i.e., not containing the reported vulnerability); for vulnerabilities with severity high and medium. Data from Decan et al.<sup>381</sup> code

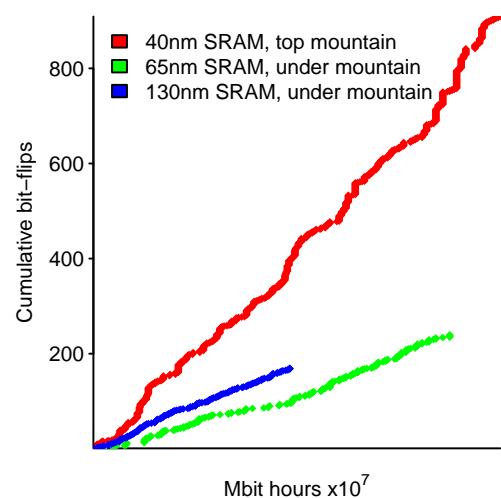


Figure 6.33: Number of bit-flips in SRAM fabricated using various processes... Data kindly provided by Autran.<sup>73</sup> code

which increases with altitude and is 10 times greater in Denver, Colorado), it has an MTBF of  $1000 \times 10^{-15} \times 4.096 \times 10^9 \times 8 = 3.2 \times 10^{-2}$  hours, around once every 33 hours. Soft errors are a regular occurrence for installations with hundreds of terabytes of memory.<sup>744</sup>

The Cassini spacecraft experienced an average of 280 single bit memory errors per day<sup>1527</sup> (in two identical flight recorders containing 2.5G of DRAM). The rate of double-bit errors was higher than expected (between 1.5 and 4.5%) because the incoming radiation had enough energy to flip more than one bit.

Uncorrected soft errors place a limit on the maximum number of computing nodes that can be usefully used by one application; at around 50,000 nodes, a system would spend half its time saving checkpoints and restarting from previous checkpoints after an error occurs.<sup>1331</sup>

Error correcting memory reduces the probability of an uncorrected error by several orders of magnitude, but with modern systems containing terabytes the probability of an error adversely affecting the result is high.<sup>744</sup> The Cray Blue Waters system at the National Center for Supercomputing Applications experienced 28 uncorrected memory errors (ECC and Chipkill parity hardware checks corrected 722,526 single bit errors and 309,359 two-bit errors, a 99.995% success rate).<sup>399</sup> Studies<sup>1061</sup> have investigated assigning variables deemed to be critical to a subset of memory that is protected with error correcting hardware, along with various other techniques.<sup>989</sup>

Calculating the FIT for processors is complicated.<sup>952</sup>

Redundancy can be used to continue operating after experiencing a hardware fault, e.g., three processors performing the same calculation, and a majority vote used to decide which of the outputs to accept.<sup>1688</sup> Software only redundancy techniques include having the compiler generate, for each source code sequence, two or more independent machine code sequences<sup>1323</sup> whose computed values are compared at various check points, and replicating computations across multiple cores<sup>1703</sup> (and comparing outputs). The overhead of duplicated execution can be reduced by not replicating those code sequences that are less affected by register bit flips<sup>486</sup> (e.g., the value returned from a bitwise AND that extracts 8 bits from a 32-bit register is 75% less likely to deliver an incorrect result than an operation that depends on all 32 bits). Optimizing for reliability can be traded off against performance,<sup>1093</sup> e.g., ordering register usage such that the average interval between load and last usage is reduced.<sup>1683</sup>

Developers don't have to rely on compiler or hardware support, reliability can be improved by using algorithms that are robust in the presence of *faulty* hardware. For instance, the traditional algorithms for two-process mutual exclusion are not fault tolerant; a fault tolerant mutual exclusion algorithm using  $2f + 1$  variables, where a single fault may occur in up to  $f$  variables is available.<sup>1107</sup> Researchers are starting to investigate how best to prevent soft errors corrupting the correct behavior of various algorithms.<sup>215</sup> Bombarding a system with radiation increases the likelihood of radiation induced bit-flips.<sup>1074</sup>

The evolution of commodity cpu and memory chips appears to be heading towards cheap and unreliable products, just like many household appliances are priced low and have a short expected lifetime...

A study by Dinaburg<sup>406</sup> found occurrences of bit-flips in domain names appearing within HTTP requests, e.g., a page from the domain ikamai.net being requested rather than from akamai.net. The  $2.10^{-9}$  bit error rate was thought to occur inside routers and switches. Undetected random hardware errors can be used to redirect an access to another site<sup>406</sup>...

If all the checksums involved in TCP/IP transmission are enabled, the users visiting Facebook on average once per day and downloading 2M of Javascript per visit, gives an expected bit flip rate of once every 5 days somewhere in the world.

The impact of level of compiler optimization on a program's susceptibility to bitflips is discussed in Section 11.2.2.

Decreased reliability as a design trade-off...?

[data/Blockchain Graveyard.htm](#)

### 6.5.1 System availability

A system is only as reliable as the least unreliable critical subsystem, and the hardware on which software runs is a critical subsystem that needs to be included in any application

reliability analysis; some applications also require a working internet connection, e.g., for database access.

Before cloud computing became a widely available commercial service, companies built their own clustered computer facilities (low usage rates of such systems<sup>755</sup> is what can make cloud providers more cost effective).

The reliability of Internet access to the services provided by other computers is not yet high enough for people to overlook the possibility that failures can occur<sup>150</sup> (see example in Section 10.5.4)...

Long-running applications need to be able to recover from hardware failures, if they are to stand a reasonable chance of completing. A process known as *checkpointing* periodically stores the current state of every compute unit, so that when any unit fails, it is possible to restart from the last saved state, rather than restarting from the beginning. A tradeoff has to be made<sup>1686</sup> between frequency of checkpointing, which takes resources away from completing execution of the application but reduces the amount of lost calculation, and infrequent checkpointing, which diverts less resources but incurs greater losses when a fault is experienced. Calculating the optimum checkpoint interval<sup>351</sup> requires knowing the distribution of node uptimes; see Figure 6.34.

The Los Alamos National Laboratory (LANL) has made public, data from 23 different systems installed between 1996 and 2005,<sup>918</sup> these systems run applications that "... perform long periods (often months) of CPU computation, interrupted every few hours by a few minutes of I/O for check-pointing." Figure 6.34 shows the 10-hour binned data fitted to a zero-truncated negative binomial distribution for systems 2 and 18.

Operating systems and many long-running programs sometimes write information about a variety of events to one or more log files. One study<sup>1692</sup> found that around 1 in 30 lines of code in Apache, Postgresql and Squid was logging code; this information was estimated to reduce median diagnosis time by a factor of 1.4 to 3. The information diversity of system event logs tends to increase, with new kinds of information being added, but the writing of older information not being switched off (because it might be useful); log files have been found to contain<sup>1186</sup> large amounts of low value information, more than one entry for the same event, changes caused by software updates, poor or no documentation and inconsistent information structure within entries.

## 6.6 Checking for intended behavior

The two main methods for checking that code behaves as intended, are: analyzing the source code to work out what it does and reviewing the behavior of the code during execution (e.g., testing). Little data is available on the kinds of problems found, and the relative cost-effectiveness of the various techniques that are used.

Many software systems support a range of optional constructs and support for these may be selected by build time configuration options. When checking for intended behavior, one of the first questions will involve the versions of the system being checked. Some systems support so many options that simply checking whether all possible configurations can be built requires a huge investment of time and computing resources.<sup>649</sup>

The further along in the development process a problem is found, the more costly it is likely to be to correct it (possible additional costs include having to modify something created between the introduction of the problem and its detection, and having to recheck work). This additional cost does not necessarily make it more cost effective to detect problems as early as possible. The relative cost of correcting problems vs. detecting problems may make it more cost effective to check for problems at specific point in the development process.

The cost of correcting problems will depend on the cost characteristics of the system containing the software; developing software for a coffee vending machine is likely to be a lot cheaper than for a jet fighter because of the cost of the hardware needed to test. Data from NASA and US Department of Defense, on relative costs of fixing problems discovered in various phases of development are very large because of the very high cost of the hardware running the software systems developed for these organizations.

To reduce time and costs the checking process may be organized into different levels of granularity, not proceeding to a higher level until the lower levels are known to work: unit testing

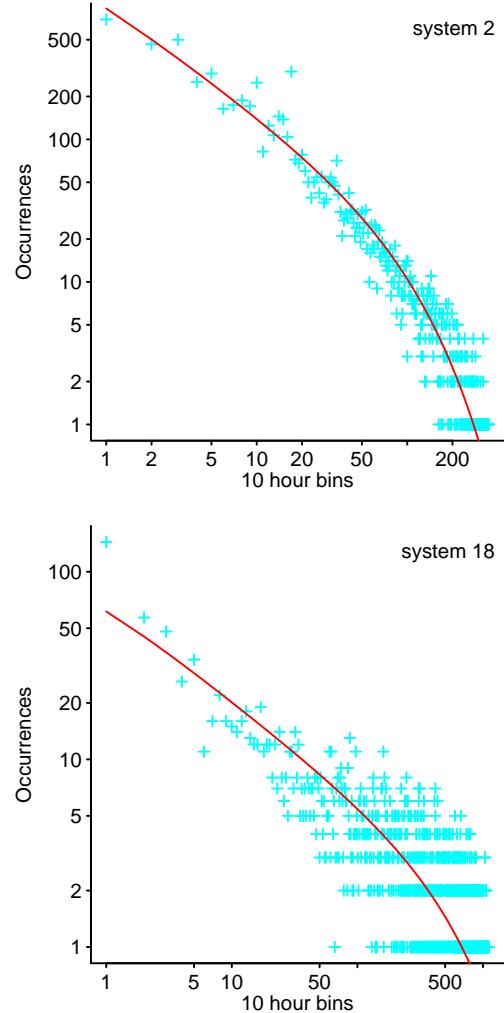


Figure 6.34: For systems 2 and 18, number of uptime intervals, binned into 10 hour intervals, red line is fitted negative binomial distribution. Data from [Los Alamos National Lab \(LANL\)](#). [code](#)

is performed by individual developers, integration testing checks that multiple components or subsystems work together and systems testing is performed on the system as a whole.

A study by Hribar, Bogovac and Marinčić<sup>729</sup> investigated *Fault Slip Through*, they analysed the development phase where a fault was found compared to where it could have been found. Figure 6.35 shows the number of faults found in various test phases (deskcheck is a form of code review performed by the authors of the code), and where the fault could have been found (as specified on the fault report; also see Antolić<sup>1582</sup>).

The problem experienced by experiments designed to compare different review methods is that the variation in skill and knowledge between the subjects, and variation in the difficulty of finding different kinds of mistakes, is often much greater than performance differences that might be attributed to the different techniques.

A study by Myers<sup>1125</sup> investigated the faults experienced using program testing and code walkthroughs/inspections. The 59 professional developers were split into two groups of 16 individual teams, who ran tests (either just with a specification, or with both a specification and program listing), and nine teams of three, who did walkthroughs/inspection... See [reliability/myers1978.R](#) todo.

A study by Finifter<sup>497</sup> investigated the mistakes found and fault experienced using manual code review and black box testing of nine implementations of the same specification. Figure 6.36 shows the number of vulnerabilities found by the two techniques in the nine implementations; some of the difference is due to the variation in the abilities and kinds of mistakes made by different developers, plus skill differences in using the programming languages.

While there has been a flurry of activity applying machine learning techniques to fault prediction, the models have not generalized outside the data used to build them (even between different versions of the same project;<sup>1714</sup> see [faults/eclipse/eclipse-pred.R](#)). Very noisy data is one problem, along with no usage data (see Section 6.1.3).

During a review, there is an element of chance in which problems individuals notice, and some problems may only be noticed by reviewers with a particular skill or knowledge.

If all reviewers have the same probability,  $p$ , of finding a problem, and there are  $N$  issues available to be found, by  $S$  reviewers, then the fraction of these issues found is:  $N(1 - (1 - p)^S)$ .

A study by Nielsen and Landauer<sup>1156</sup> investigated how the number of different usability problems discovered as the number of test subjects increased, based on data from 12 studies. Figure 6.37 shows how the number of perceived usability problems increased as the number of test subjects increased; lines show the regression model fitted by the above equation (both  $N$  and  $p$  are constants returned by the fitting process).

When the probability of finding a problem varies between reviewers, there can be a wide variation in the number of problems reported by different groupings of individuals.

A study by Lewis<sup>950</sup> resulted in 15 reviewers finding 145 usability problems. How many problems would two of these find, how many would three find? Figure 6.38 is based on the issues found by every pair, triple (etc, up to five) of reviewers. The mean value of issues found increases with review group size, as does the absolute spread of the number found. Half of all issues were only found by one reviewer, and 15% found by two reviewers.

## 6.6.1 Review meetings

Review meetings support a variety of functions, including: highlighting of important information between project members (i.e., ensuring that people are kept up to date with what others are doing) and uncovering potential problems before changing things becomes much more expensive.

Most research has used problems-found as the metric for evaluating review meetings, in particular potential fault experiences found during code reviews. Problems found is something that is easy to measure, code is readily available and developers to review it are likely to be more numerous than people with the skills needed to review requirements and design documents (which do not always exist, as such).

The range of knowledge and skills needed to review requirements and design documents may mean that some of those involved concentrate on areas that are within their domain

of expertise.<sup>435</sup> Many of the techniques used for estimating population size assume that capture sites (i.e., reviews) have equal probabilities of flagging an item, and estimates based on data from meetings where reviewers have selectively read documents will be biased; see [reliability/eickt1992.R](#).

Detecting problems may not even be the main reason for performing code reviews,<sup>80</sup> keeping teams members abreast of developments and creating an environment of shared ownership of code may be considered more important.

People tend to have a repertoire of actions, which are used on a regular basis,<sup>7</sup> and result in mistakes being introduced into the code. Recurring coding patterns associated with known mistakes can be searched for, in code, and flagged. Depending on the construct the search process may best be performed by a tool or by a developer reviewing the code.

Ideally code flagged by a tool is incorrect (e.g., reading from an uninitialized variable), but the analysis performed may not be sophisticated enough to handle all possibilities<sup>1378</sup> (e.g., there is some uncertainty about whether the variable being read from has been written to), or the usage may simple be suspicious (e.g., use of assignment in the conditional expression of an if-statement, when an equality comparison was intended, i.e., = had been typed instead of ==). The issue of how developers might respond to false positive warnings is discussed in Section 9.1.

A study of one tool<sup>1707</sup> found a strong correlation between flagged code and faults experienced during testing and faults reported by customers (after the output of the tool had been cleaned by a company specializing in removing false positive warnings from static analysis tool output).

Traditionally a human code review (other terms include *code inspection* and *walkthroughs*<sup>525</sup>) has involved one or more people reading another developer's code, and then meeting with the developer to discuss what they have found. These days the term is also associated with maintainers reviewing code that has been pushed to a project's version control system, to check that it is ok to merge...

A variety of different code review techniques have been proposed, including: Ad-hoc (no explicit support for reviewers), Checklist (reviewers work from a list of specific questions that are intended to focus attention towards common problems), Scenarios-based (each reviewer takes on a role intended to target a particular class of problems), and Perspective-based reading (reviewers are given more detailed instructions, than are given in Scenario-based reviews, about how to read the document; see Section 13.2 for an analysis)... .

A study by Hirao, Ihara, Ueda, Phannachitta and Matsumoto<sup>704</sup> investigated the impact of positive and negative code reviews on patches being merged or abandoned (for Qt and OpenStack). A logistic regression model found that for Qt positive votes were more than twice as influential, on the outcome, as negative votes, while for, OpenStack negative votes were slightly more influential (see [reliability/OSS2016.R](#)).

A study by Porter, Siy, Mockus and Votta<sup>1264</sup> recorded code inspection related data from a commercial project over 18 months (staffed by six dedicated developers, and five developers who also worked on other projects). The best fitted regression model had the number of mistakes found being proportional to the log of the number of lines reviewed, and the log of meeting duration; this study is discussed in Section 13.2, also see Figure 11.32.

Job status of code reviewer...?

## 6.6.2 Testing

The purpose of testing is to gain some level of confidence that the software operates in a way that is likely to be acceptable to the customer, given how the vendor intends it to perform (e.g., meets acceptance payment criteria or is unlikely to experience a high rate of customer returns).

Most published research of testing has been at the system level...

During testing, one indication that the software is becoming more reliable, is that the number of previously unseen faults experiences per unit of test effort is decreasing; another reason for a decrease in new fault discovery is replacement of existing testers by less skilled staff. The extent to which the reliability, as experienced by the customer, increases depends on the overlap between test inputs and likely customer inputs to the software.

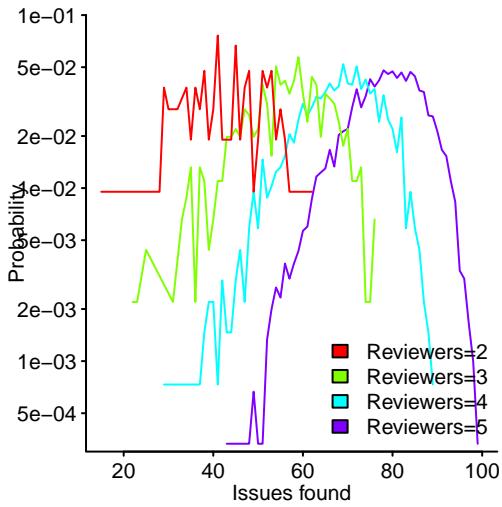


Figure 6.38: Probability (y-axis) of a given number of issues being found (x-axis), by groups containing a given number of people (colored lines). Data from Lewis.<sup>950</sup> [code](#)

A study by Stikkel<sup>1509</sup> investigated three industrial development projects. Figure 6.39 shows the (normalised) number of faults discovered per man-hour of testing, averaged over a week, for these projects; two show a sharp decline in new fault discoveries, per man-hour, in the closing weeks of testing, while the third shows no such pattern.

The number of possible combinations of inputs to a non-trivial program is so large, that by comparison, the number of tests that are practical to run is insignificant. However, the number of input combinations used by customers may be small enough to be make thorough testing viable, but the detailed information on the input combinations used by customers is often unavailable.

An example of how the input used for random testing can be unrepresentative of customer input is provided by a study<sup>299</sup> that performed random testing of the Eiffel base library. The functions in this library contain extensive pre/post condition checks, and random testing found twice as many mistakes in these checks as the implementation of the functionality; the opposite of the pattern seen in user reported faults.

Sources of manually created tests include developers packaging up tests they have written to check their own code, fault reports submitted by users and in some cases third-party written test suites. Some customers are sufficiently interested in the behavior of the software they buy, they are willing to fund the development of a test suite. The US Government appreciated the economic advantages of being able to select from multiple hardware vendors, and ensuring that compilers from different vendors could compile the same source code was an important factor in increasing program portability; validation suites were funded for Cobol and Fortran,<sup>1187</sup> and later SQL, POSIX<sup>2</sup> and Ada.<sup>5</sup>

Manual tests can be very effective, but creating them is very time-consuming<sup>?, 1187</sup> and expensive. Various kinds of automatic test generation are available.

Testing that the behavior of a program is as intended requires knowledge of the intended behavior, for a given input. While some form of automatic input generation is possible, in only a few cases<sup>46</sup> is it possible to automatically predict the expected output from the input, independently of the software being tested. One form of program behavior is easily detected, abnormal termination, and some forms of fuzz testing use of this as their correctness criteria.

When multiple systems supporting the same functionality are available, it may be possible to use differential testing to compare the outputs produced from a given input (a difference being a strong indicator that one of the systems is behaving incorrectly)<sup>282</sup>...

While studies<sup>891</sup> have found that the majority of faults are experienced with test cases involving a change of one input value, this result may be due to most test cases involving a single input value, or some other reason (the necessary information is rarely provided)...

ISO Standards have been published covering methods for measuring conformance to particular standards<sup>760, 761</sup> and requirements for test laboratories.<sup>758</sup> However, the lack of commercial interest...

Tests may need to be updated when code changes, and new tests written to handle any new modified behavior; maintenance can be a significant expense.<sup>?</sup> Like production code, tests can contain mistakes.<sup>1583</sup>

An analysis of a few projects<sup>1694</sup> found no correlation between the growth of project and test code (see [time-series/argouml\\_complete.R](#)).

To what extent do test suites change over time? A study by Marinescu, Hosek and Cedar<sup>1019</sup> measured the statement and branch coverage (using the test suite distributed with the program's source) of six open source programs over time. Figure 6.40 shows that for some widely used programs to statement coverage of the test suite did not vary much over five years.

Evolution of testing library usage...?

### 6.6.2.1 Combinatorial testing

emailed for data...? emailed - received...?

?

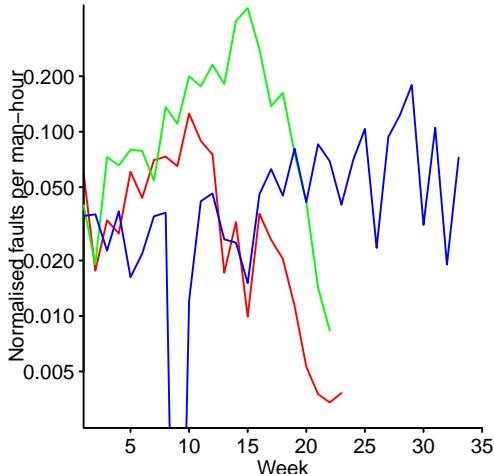


Figure 6.39: Number of faults experienced per unit of testing effort, over a given number of weeks. Data from Stikkel.<sup>1509</sup> [code](#)

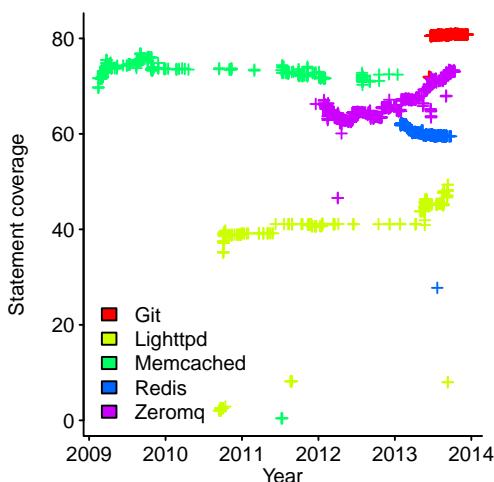


Figure 6.40: Statement coverage achieved by the respective program's test suite (data on sixth program was not usable). Data from Marinescu et al.<sup>1019</sup> [code](#)

### 6.6.2.2 Beta testing

The people who use software systems may operate it using input profiles that are very different from those envisaged by the developers who tested the software. Beta testing is a way of discovering problems with software (e.g., coding mistakes and incomplete requirements), when processing the input profiles of the intended users. The number of problems found during beta testing, compared to internal testing provides feedback on the relevance of the usage profile that drives the test process.<sup>1015</sup>

Behavior during beta testing different from after release... Figure are images :-?

### 6.6.2.3 Estimating test effectiveness

Is the test process for a software system able to reliably provide the desired level of confidence that the software is good enough to be acceptable to the customer?

A necessary requirement for checking the behavior of code is to execute it, every statement not executed is untested. Various coverage criteria are available, for instance percentage of program statements or branches executed by the test process (the coverage achieved by a test suite is likely to vary between platforms and even compiler used<sup>544</sup> more???).

A study by Inozemtseva and Holmes<sup>750</sup> investigated test coverage of five very large Java programs. The results showed a consistent relationship between percentage statement coverage,  $sc$ , and percentage branch coverage,  $bc$  (i.e.,  $bc \propto sc^{1.2}$ ), and percentage modified condition coverage,  $mc$  (i.e.,  $mc \propto sc^{1.7}$ ); see [reliability/coverage\\_icse-2014.R...](#)

Software written for safety critical applications often has to have 100% MC/DC coverage<sup>289</sup>... no data found...

A study by Gopinath, Jensen and Groce<sup>598</sup> investigated some of the characteristics of coverage metrics for the test suites of 1023 Java projects. Figure 6.41 shows fraction of statement coverage against branch coverage; each circle is data from one project. The lines are fitted regression models, which contain a strong interaction between coverage and log KLOC... no correlation between log(loc) and test suite coverage...

A study by Kang, Ray and Jana<sup>823</sup> investigated the number of statements contained along the execution paths, within a function, followed after a call to a function that could return an error, i.e., the error and non-error paths. Figure 6.42 shows that non-error paths often contained more statements than the error paths...

One explanation for the slow initial growth in branch coverage seen in Figure 6.41 is that the test suites primarily contain positive tests (which contain more statements)... ?

A study by Čaušević, Shukla, Punnekkat and Sundmark<sup>1577</sup> found that developers wrote almost twice as many positive tests as negative tests, for the problem studied; see [reliability/3276-TDD.R.](#)

Decision points in code (i.e., if statements) select the next basic block to execute; using basic blocks as a coverage metric throws away information on the length of statement sequences. The regression fit in Figure 6.43 shows a linear relationship between basic block coverage and decision coverage; the expected relationship.

Growth in coverage as the number of tests increases will depend on the distribution of input values contained in the tests and the characteristics of the code being tested.

A study by McAllister and Vouk<sup>1033</sup> investigated the coverage of 20 implementations of the same specification. Two sets of random tests were generated using different selection criteria and 796 tests designed to provide full functional coverage of the specification. Figure 6.44 shows the fraction of basic blocks covered as the number of tests increases, for the 20 implementations (same color), and three sets of tests (different colors); the lines are fitted regression models of the equation:  $coverage_{BB} = a \times (1 - b \times \log(tests))^c$ , where:  $a$ ,  $b$  and  $c$  are constants ( $c$  is between -0.35 and -1.7 for this example).

promised for data...?

Many software systems support multiple configurations and running the same tests on different configurations can result in different sets of statements being executed;<sup>1296</sup>

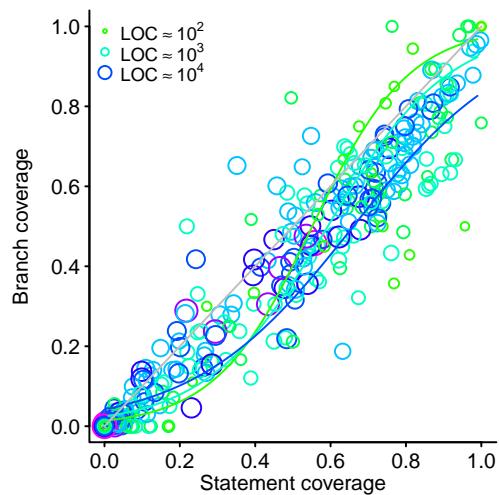


Figure 6.41: Statement coverage against branch coverage for 300 or so Java projects; colored lines are fitted regression models for three program sizes, equal value line in grey. Data from Gopinath et al.<sup>598</sup> [code](#)

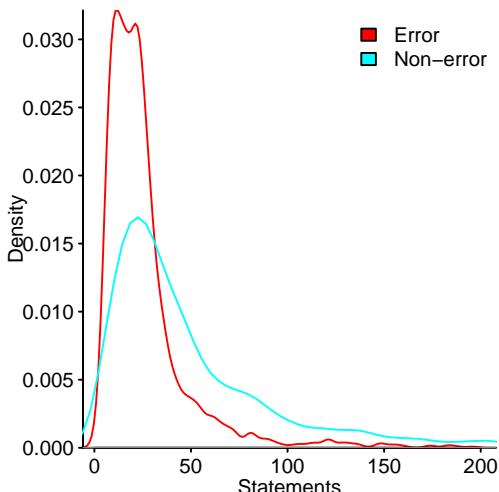
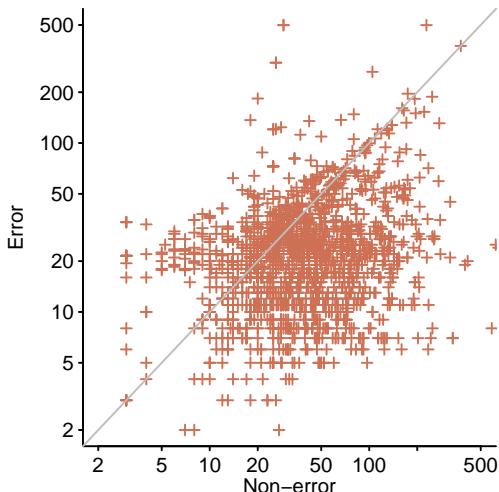


Figure 6.42: Number of statements executed along error and non-error paths within a function. Data kindly provided by Kang.<sup>823</sup> [code](#)

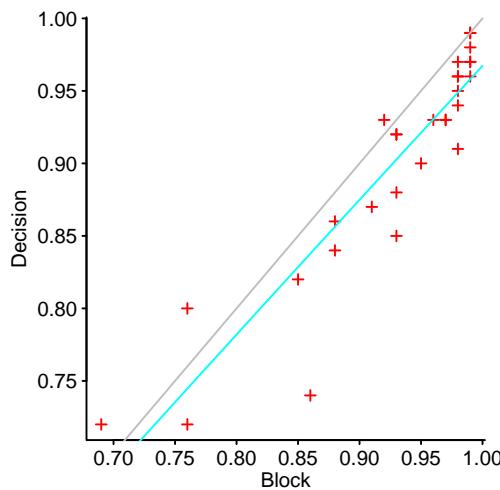


Figure 6.43: Basic block coverage against branch coverage for a 35 KLOC program. Data from Gokhale et al.<sup>582</sup> code

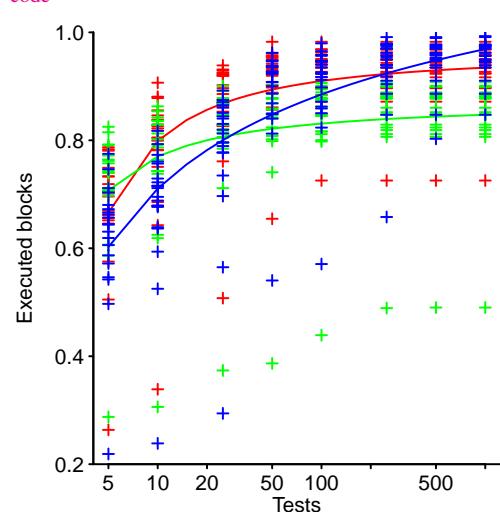


Figure 6.44: Fraction of basic blocks executed by a given number of tests, for 20 implementations using three test suites. Data from McAllister et al.<sup>1033</sup> code

Another technique for estimating the effectiveness of a test suite for detecting coding mistakes is to introduce known mistakes into the source and measure the percentage detected by the test suite, i.e., they cause a change of behavior that is detected by the test process. The modified source is known as a *mutant*. Ideally the characteristics of the mutations matches the characteristics of the mistakes made by the developers writing the code (mutant generation techniques tend to be generic and generated mistakes have been found not to have the characteristics of developer coding mistakes<sup>599</sup>). A test suite that kills a high percentage of mutants (what self-respecting developer would ever be happy just detecting mutants?) is considered to be more effective than one that kills a lesser percentage.

Figure 6.45 shows statement coverage against percentage of mutants killed. The lines are fitted regression models, which contain a strong interaction between coverage and log KLOC... no correlation between  $\log(loc)$  and test suite coverage...

A test suite's mutation score converges to a maximum value, as more mutants are tested. For programs containing fewer than 16K executable statements,  $E$ , the number of mutants needed has been found to grow no faster than  $O(E^{0.25})$ <sup>1700</sup>, a worst case confidence interval for the error in the mutation score can be calculated.<sup>597</sup>

There have been a few studies investigating the coverage of compiler validation suites<sup>296, 783</sup>...

A statement might only fail to behave as intended when the input has specific characteristics...

### 6.6.3 Cost of testing

Minimising costs by reducing the number of tests that need to be run...

One metric sometimes used as an indication of changes in the reliability of a software system is the number of faults discovered, per unit effort...

A study by Do, Mirarab, Tahvildari and Rothermel<sup>409</sup> investigated the cost benefit trade-offs associated with the cost of regression testing, time to market and cost of faults reported by customers...

When to stop testing software? Stopping conditions are discussed in Section 13.2.4...

It is possible to derive a relationship for the cost/benefit of continued testing vs. the cost of fixing faults experienced by customers... Stopping rules for testing after a given amount of time, based on number of faults experienced...<sup>275, 1685</sup>

### 6.6.4 Runtime issues

Techniques that have been used to mitigate against the occurrence of an error that occurs during program execution include:...

If multiple developers create implementations based on the same specification, will coding mistakes made by each developer, in each program, be independent of each other? Known as *N-version programming*

?

?

?

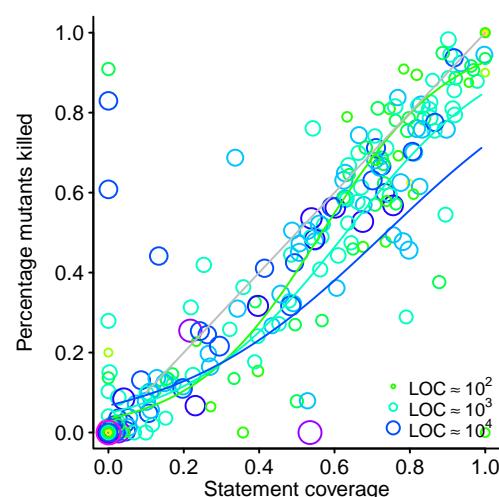


Figure 6.45: Statement coverage against mutants killed for 300 or so Java projects; colored lines are fitted regression models for three program sizes, equal value line in grey. Data from Gopinath et al.<sup>598</sup> code

# Chapter 7

## Source code

### 7.1 Introduction

Source code is a form of communication, written in a programming language, whose purpose is to communicate a sequence of actions or required results<sup>i</sup> to a computer or a person.

Building a software system involves arranging source code in a way that causes the desired narrative to emerge as the program is executed with the appropriate inputs.

This chapter discusses source code from two perspectives: reducing the cognitive effort developers need to invest in processing code and how patterns of source usage can be used to validate theories of developer behavior.

People are inveterate storytellers and narrative is another way of interpreting source code...

There are many technical similarities between programming languages and human languages, however, the ways in which they are used to communicate are very different (differences have also been seen in medical imaging of brain activity<sup>512</sup>). There has been a lot of research investigating the activities involved in processing written representations of human language<sup>360</sup> (i.e., prose<sup>ii</sup>; see Section 2.3), but few scientific studies of the activities involved in processing source code. For instance, eye tracking is commonly used to study reading prose, but is only just starting to be used to study reading code (see Figure 2.16). This chapter draws on the results from studies of prose, and attempts to highlight patterns of behavior that might apply to source.

Readers may be familiar with an approach to studying language based on the products of language (e.g., words and sentences) abstracted away from their use (e.g., the work of Chomsky). The cognitive linguistics<sup>1304</sup> approach is based around how people use language, with context taken into account (e.g., intentions and social normals, such as the work of Grice<sup>625</sup>). This chapter takes a cognitive linguistics approach to source code.

- Human language use is a joint activity, based on common ground between speaker and listener<sup>iii</sup>. What a speaker says is evidence a listener uses to create an interpretation of the speaker's intended meaning. A conversation involves cooperation and coordinating activities...

Grice's maxims<sup>625</sup> provide a model of human communication, involving speaker aims and listener assumptions. Perhaps the most important aspect of human language communication is the assumption of relevance<sup>300, 1476</sup> (and even optimal relevance). A speaker says something because they believe it is worth the effort, with both speaker and listener assuming that what is said is relevant to the listener and worth the listener investing effort to understand.

---

<sup>i</sup> In a declarative language, such as SQL, the intended result is specified (e.g., return information matching the specified conditions), while code written in an imperative language specifies actions to be performed (with the author being responsible for sequencing the actions to produce an intended result).

<sup>ii</sup> English prose has been the focus of most studies, with prose written in other languages not yet so extensively studied.

<sup>iii</sup> "vaj laH: pejatlh lion ghaH yajbe' maH." Wittgenstein.

- Communicating using source code is a take it or leave it transaction, what the speaker *said* is treated as definitive by the listener, intent is not part of the process. There is no cooperative activity, and the only coordination involves the speaker modifying what has been *said* until the desired listener response is achieved.

Creating a program requires explaining everything that is relevant, including all the supporting background detail (commonly occurring narrative subplots are often supplied by library functions).

Computers do not have a conception of application assumptions and relevance. Experienced developers are aware of this and adjust their approach to communication; novices have to learn how to communicate with a listener who is not aware of their intent. One aspect of learning to program is learning to communicate with a subject who will not make any attempt to adapt to the speaker; patterns have been found in the ways novices misunderstanding the behavior of code.<sup>1224</sup>

Source code is the outcome of choices made in response to the application domain problem being solved, the algorithms used, implementation requirements, functionality provided by the programming language and the beliefs, experiences and preferences of the developers writing the code, along with a desire for short-term gratification.<sup>524</sup>

Research on application domain driven patterns of usage or high-level patterns present in programs has been thin on the ground. In the domain of compiler writing, the production-quality compiler-compiler project,<sup>947</sup> in the late 1970s, created a way of organizing optimizing compiler that has been widely used ever since.

The low-level patterns of usage found in source code arise from the many developer coding choices made in response to immediate algorithmic and implementation requirements. High-level problems need to be broken down into smaller problems, and so on down, until the developer recognises how a solution can be composed using code.

For instance, implementing the  $3n + 1$  problem requires testing whether  $n$  is odd or even. In the van der Meulen study,<sup>1591</sup> the expressions used for this test include:  $n \% 2$  and  $n \& 1$  (89% and 8% of uses respectively), along with less common sequences such as:  $(n/2)*2 == n$  and  $(n >> 1) << 1 == n$ ; around 50% of the conditions were true when  $n$  was even. Over 95% of expressions appeared as the condition of an if-statement, with most the others appearing as the first operand of a ternary operator (e.g.,  $n=(n \% 2) ? 3*n+1 : n/2$ ).

Developers have a collection of beliefs, and mental models, about the semantics of the programming languages they use, as well as a collection of techniques they have grown accustomed to using...

If, after executing the two statements: `x=1;y=x+2;`, the statement: `x=3;`, is executed, is the value of  $y$  now 5? In many programming languages the value of  $y$  remains unchanged by the second assignment to  $x$ , but in reactive programming languages<sup>93</sup> (found in spreadsheet programs) statements can express a relationship, not a one time assignment of a value (novice developers have been found to give a wide variety of interpretations of the assignment operator<sup>182</sup>).

While spreadsheets support the specification of formula and relationships between memory locations, they have not traditionally been treated as part of software engineering. One reason has been the lack of large samples of usage to study; legal proceedings against large companies are helping to change this situation.<sup>687</sup>

Acquiring an understanding of the behavior of a program, by reading its source code, is not an end in itself; one reason for making an investment to acquire this understanding, is to be able to predict its behavior sufficiently well to be able to change it. By reading source code, developers acquire beliefs about it, which are a means to an end; *understanding a program* is a continuum, not a yes/no state.

Source code is used to build programs and libraries (or packages). There are a variety of different kinds of text that might be referred to as *source code*, e.g., text that is compiled to produce an executable program, text that is used to direct the compilation process (such as Makefiles and shell scripts), text that resembles prose in configuration files...

The complexity and inter-connectedness found in software systems can also be found in other ecosystems. A study by Braha and Bar-Yam<sup>196</sup> investigated the network connections in a 16-story hospital, pharmaceutical facilities design, a General Motors vehicle design facility and

	Nodes	Edges	Average path length	Clustering coeff	Degree	Density
<b>Hospital</b>	889.00	8178.00	3.12	0.11	18.40	0.02
<b>Pharmaceutical</b>	582.00	3689.00	2.63	0.12	12.68	0.02
<b>Software</b>	466.00	1245.00	3.70	0.14	5.34	0.01
<b>Vehicles</b>	120.00	417.00	2.88	0.13	6.95	0.06

Table 7.1: Values of various attributes of the communication network graphs in various organizations. Data from Braha et al.<sup>196</sup> [code](#)

Linux. Table 7.1 lists the values of various properties of the networks and from a component network perspective software looks middle of the road.

A study by German, Adams and Hassan<sup>560</sup> investigated how developers working on the Linux kernel used Git...

?

### 7.1.1 Quantity of source

What is the size distribution of software systems?

A study by Landman, Serebrenik, Bouwers and Vinju<sup>911</sup> measured 13,103 Java projects. Figure 7.1 shows the number of source files, methods and sloc (within each method) for each project. A fitted regression model finds that a projects' total source lines of code is well fitted by the equation:  $sloc = e^{1.98} \text{methods}^{1.12} \text{files}^{-0.13}$

See Figure 5.4 and Figure 5.1 for information on time and costs for developing programs.

Lines of code, versions, files, support files, bytes in system shipped...

In many languages a *method* (also known as a *function*, *procedure* or *subroutine*) is the smallest self-contained unit of source code.

A study by Landman, Serebrenik, Bouwers and Vinju<sup>911</sup> measured 19K open source Java projects, and the 9.6K packages (written in C) contained in a Linux distribution. Figure 7.2 shows the number Java methods and C functions containing a given number of source lines. A power law provides a good fit to both sets of data; see [sourcecode/Landman\\_m\\_ccsloc.R](#).

If we assume the projects implemented the same range of applications to equivalent levels of functionality, then differences in the distribution of method/function sizes is the result of developer use of language features, e.g., constructs that are supported and common coding idioms.

A study by Herraiz measured the number of files and SLOC in a snapshot of 3,781 open source projects. Figure 7.3 shows number of files against LOC, as a density plot;  $LOC = 161 * File^{0.99}$  is a fitted regression model. Another study<sup>689</sup> of 13 Open source projects found that the number of files was a good predictor of the number of lines of code (the multiplying factor varied from 100 to 565)...

Sometimes the only information available about a program is contained in its executable form; there have been studies<sup>1521</sup> that have built models estimating the length of the original source from the compiled machine code (the language in which code is written can have a large impact on machine code characteristics<sup>239</sup>).

The relationship between static and dynamic counts of machine code, compiled from the same source code, is affected by the processor characteristics and compiler used; Figure 7.4 shows static and dynamic percentage of call instruction opcodes, in the compiled form of various C programs (targeting various processors).

The quantity of source has been found to be an approximate predictor of compile time (e.g., one study<sup>1081</sup> of Ada compilers found compile time was proportional to the square root of the number of tokens; see [sourcecode/ADA177652.R](#)). A study by Auler and Borin<sup>69</sup> investigated the time taken by a JIT compiler to generate code for functions from the SPEC CPU2006 benchmark. Figure 7.5 shows the time taken to generate machine code for 71,200 functions, containing a given number of LLVM instructions (an intermediate code). The two trends in the data are: compile time not depending on function size and compile time increasing linearly (once functions contain more than 100 instructions).

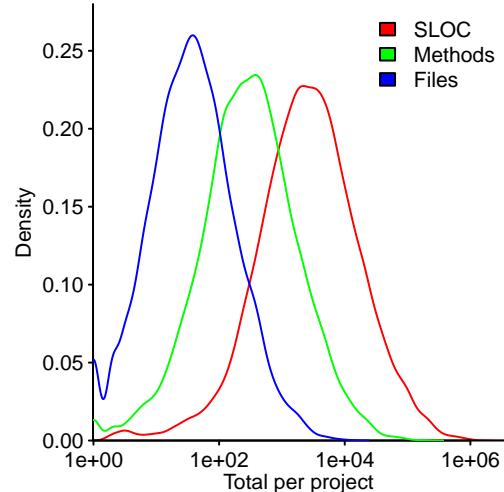


Figure 7.1: Number of source files, methods and lines of code, within methods, contained in each of 13,103 Java projects; lines are kernel density plots. Data kindly provided by Landman.<sup>911</sup> [code](#)

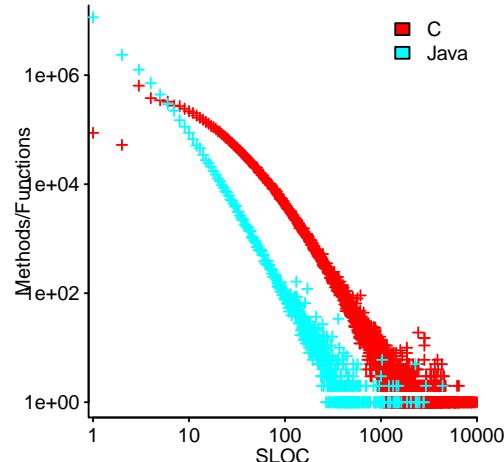


Figure 7.2: Number of methods/functions containing a given number of source lines; 17.6M methods, 6.3M functions. Data kindly provided by Landman.<sup>911</sup> [code](#)

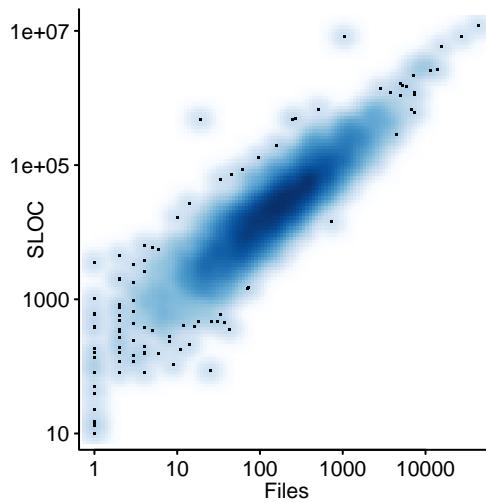


Figure 7.3: Number of files and lines of code in 3,782 projects hosted on Sourceforge. Data from Herraiz.<sup>1529</sup> code

Many software systems allow some features to be optionally selected at build time. One technique for selecting which features to include in a program is conditional compilation, e.g., `#if/#endif` in C and C++. A study by Liebig, Apel, Lengauer, Kästner and Schulze<sup>958</sup> measured various attributes associated with the use of conditional compilation directives in 40 programs written in C (header file contents were ignored); Figure 7.6 shows how the number of optional features increases with lines of code.

?

## 7.2 Desirable characteristics

The possibility that, at some point in the future, new code is added to existing code, or existing code is modified, provides an investment opportunity to reduce future costs by using code that has the appropriate characteristics.

- a regular turnover of staff (see Section 3.3.2) creates an incentive to reduce upfront investment in source code familiarisation costs, e.g., using constructs whose behavior is the same across implementations means that recruitment is not restricted to developers with experience of a specific implementation (this aim has to be weighed against the benefits provided by implementation specific features),
- developers' primary interaction with source code is reading it to obtain information and integrating new functionality into existing code. Increasing the memorability of previously read code reduces the time needed to reread it and reduces the cognitive effort needed to understand the behavior...
- efficiency... inefficient SQL queries<sup>204</sup> ...

Questions about what code is likely to change is usually a question about what functionality is likely to change, or be added.

The terms such as *Maintainability*, *readability* and *testability* are often cited as being desirable characteristics. However, their definition is often shaped to fit the research idea or product functionality being promoted, and appear with easier, cheaper, quicker; these phrases are essentially meaningless marketing hooks.

Books, reports and memos containing recommendations about how code should be written and language constructs to avoid, so called *coding guidelines*, are something of a cottage industry, e.g., for C.<sup>350,523,534,660,725,767,835,872,1037,1090,1256,1257,1297,1301,1308,1402,1477,1481,1513</sup>

Stylistically, these documents are often more akin to literary criticism than engineering principles, i.e., they express personal opinions that are not derived from evidence (other than perhaps personal experience). Recommendations against the use of particular language constructs may be based on the construct being repeated involved in reported faults; however, the fault proneness of alternative constructs, that might be used are rarely included in the analysis, i.e., the current usage may be the least bad of the available options. C preprocessor,<sup>1058</sup> ...

Developers use language constructs because they are perceived to provide a worthwhile benefit. While it might be possible to choose between several language constructs to implement some functionality, at the time of writing there is no available evidence showing that any construct is more likely to have some desirable property, compared to another (e.g., less error prone, easier to modify or to understand by future readers of the code).

A study by Korson and Vaishnavi<sup>878</sup> investigated the impact of modularity on the time taken to modify a program... [developers/korson\\_87.R](#)

??

### 7.2.1 Programming languages

Do some programming languages require more, or less, effort from developers, to write code that has desirable characteristics?

Every programming language has its fans, people who ascribe various positive qualities to programs written in their favorite language, or in languages supporting a particular characteristic (e.g., strong typing). There is little experimental evidence for language characteristics

having an impact on developer performance, and even less evidence for specific language features having a performance impact.

Factors that might generate a measurable difference in developer performance, when using different programming languages, include: individual knowledge and skill using the language, familiarity with the application domain, and interaction between the programming language and problem, e.g., it may be easier to write a program to solve a problem using language X than using language Y.

Learning... Having subjects solve the same problem multiple times creates a learning opportunity, allowing subjects to produce an improved performance on each implementation. The experimental procedures employed generates information that can be used during the analysis of the data to factor this ordering effect into the created model.

Studies that compare languages using small programs<sup>1135, 1591, 1716</sup> suffer from the possibility of a strong interaction between the problem being solved, and the available language constructs (an effect that is likely to diminish as the size of the system grows). A study using small programs will fail to detect differences cause by differing support for programming in the large...

? languages used together, known by same person, emailed...

A realistic comparison of two languages requires information from many implementations of large systems targeting the same application domain.

A study by Waligora, Bailey and Stark<sup>1627</sup> compared various lines of code measurements (e.g., declarations, executable statements and code reuse) of 11 Ada and 19 Fortran NASA ground station software systems. While there were differences in patterns of behavior for various line counts, these differences could have been caused by factors not discussed in the report (e.g., the extent to which individual companies were involved in multiple projects and so in a good position to evaluate the potential for reuse of code from previous projects, or the extent to which project requirements specified that code should be written in a way likely to make it easier to reuse; see [projects/nasa-ada-fortran.R](#)).

Difference in performance between subjects and learning effects can dominate studies based on either small programs or experiments run over short intervals.

A study by Prechelt and Tichy<sup>1280</sup> investigated the impact of parameter checking of function calls (when the experiment was performed, C compilers that did not perform any checking on the arguments passed to functions, so called K&R style, were still in common use). All subjects wrote two programs, for one program using a compiler that performed the function call argument checking and for the other used a compiler that did not perform this checking. Subjects were randomly assigned to the problem to solve first, and the compiler to use for each problem. The time to complete a correct program was measured.

Fitting a regression model shows that the greatest variation in performance occurred between subjects (standard deviation of 74 minutes), the next largest effect was problem ordering (with the second problem being solved 38 minutes, on average, faster than the first). The performance improvement attributed to argument checking is 12 minutes, compared to no argument checking (see [experiment/tcheck98.R](#)).

Proponents of strongly typed programming languages claim that using such languages provides benefits (e.g., fewer mistakes appear in the delivered software), compared to using languages that are not strongly typed. The following are studies for which data is available:

- Gannon.<sup>543</sup> Implemented compilers for two simple languages (think BCPL and BCPL+a string type and simple structures; by today's standards both languages were weakly typed, but one less so than the other). One problem was solved by subjects, this was designed to require the use of features available in both languages, e.g., a string oriented problem (final programs were between 50-300 lines). The result data included number of errors during development and number of runs needed to create a working program (this happened in 1977, before the era of personal computers, when batch processing was king; see [experiment/Gan77.R](#)).

There was a small language difference in number of errors/batch submissions; the difference was about half the size of that due to the experimental order of language use by subjects, both of which were small in comparison to the variation due to subject performance differences. While the language effect was small, it exists. To what extent can the difference be said to be due to stronger typing rather than only one language having built in support for a string type?

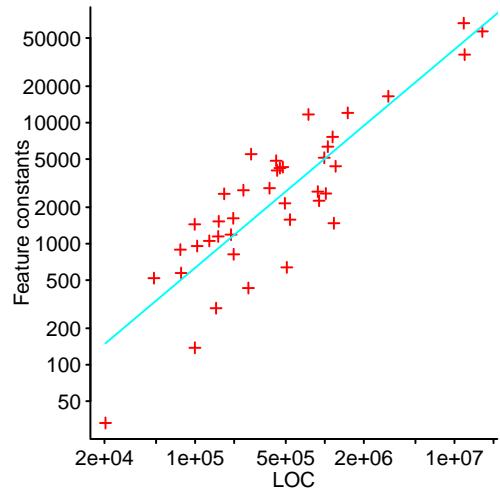


Figure 7.6: Number of *feature constants* against LOC for 40 C programs, with fitted regression line. Data from Liebig et al.<sup>958</sup> code

- Mayer, Kleinschmager & Hanenberg:<sup>865, 1029</sup> Two experiments using different languages (Java and Groovy) and multiple problems; the performance metric was time for each subject to complete the task. There was no significant difference due to just language, but large differences due to language/problem interaction with some problems solved more quickly in Java and other more quickly in Groovy and learning took place (i.e., the second task was completed in less time than the first). As always, large variations between subjects (see [experiment/mayerA1-oopsla2012.R](#) and [experiment/kleinschmagerA1.R](#)).
- Hoppe and Hanenberg:<sup>722</sup> Used one language (Java) and multiple problems involving making use of either Java's generic types or non-generic types. Again, the only significant language difference effects occurred through interaction with other variables in the experiment (e.g., the problem or the language ordering) and there were large variations in subject performance.

To summarise: when a language typing/feature effect has occurred its contribution to overall developer performance has been small. Possible reasons for small or non-existent, include:<sup>iv</sup> the use of subjects with little programming experience (i.e., students; experienced developers are more likely to make full use of the consistency checking provided by a type system), and small size of the programs (type checking comes into its own when used to organize and control large amounts of code).

Studies<sup>1085</sup> investigating the use of a human language, to specify solutions to problems, have found that subjects make extensive use of the contextual referencing that is common in human communication. This usage, and other issues, make it extremely difficult to automatically process the specified instructions.

## 7.2.2 Understanding code

Understanding (the term *comprehension* is used in prose related research and both are used in software engineering research papers), in the context of source code, involves finding a good-enough meaning of source code to complete a given task. The process can involve at one or more levels of abstraction, e.g., a sequence of lines of code, methods in a file, classes in a program... Developers do not understand programs, they learn about them; a continuous process involving the creation of new beliefs and the modification of existing ones, with no well-defined ending.

The cost-effectiveness of understanding the complete program is likely to decrease as the size of the program increases; a study<sup>974</sup> based around a 250-line Fortran program found some developers using an as-needed strategy and others attempting to understand the complete program.

One study<sup>1231</sup> found that developers with the best performance answering questions about the code they had studied, had built an application domain model, from the code (building a domain model would require some familiarity with the application domain addressed by the 200-line program used in the study).

The beliefs acquired are influenced by existing beliefs about the programming language it is written in, general computing algorithms, and the application domain.<sup>1410</sup>

Developers sometimes refer to the *intended meaning* of source code, i.e., the meaning that the original author of the code intended to implement. Code comprehension being an exercise in obtaining an intended meaning that is assumed to exist. However, the only warranted assumption that can be made about source code is that the operations specified in it contribute to a meaning.

What can be done to reduce the effort needed to find a good-enough interpretation of code and to increase the likelihood that this information can be accurately recalled later?

Activities that appear to be very complicated, can have a simple description. For instance, some hunting rituals intended to select the best hunting location are randomization algorithms,<sup>1412</sup> whose social impact is to reduce the likelihood of over-hunting any location.

A study by Bransford and Johnson<sup>202</sup> investigated the impact of a top-level description on the amount of information subjects' remembered about a task. Try to give a meaning to the task described in the outer margin, while remembering what is involved (taken from the study):

---

<sup>iv</sup> Your author declares his belief that when integrated into the design process, strong typing has cost/benefit advantages.

Table 7.2 shows that subjects' recalled over twice as much information, if they were given a meaningful phrase (the topic), before reading the passage. The topic of the passage above is *answering questions*.

	No Topic Given	Topic Given After	Topic Given Before	Maximum Score
Comprehension	2.29 (0.22)	2.12 (0.26)	4.50 (0.49)	7
Recall	2.82 (0.60)	2.65 (0.53)	5.83 (0.49)	18

Table 7.2: Mean comprehension rating and mean number of ideas recalled from passage (standard deviation in parentheses). Adapted from Bransford and Johnson.<sup>202</sup>

?

The complexity of human languages, which have to be processed in real-time while listening to the speaker, is constrained by the working memory capacity of those involved in the communication activity.<sup>564, 669</sup> The capacity limits that make it difficult for speakers to construct complicated sentences, in real-time, create a benefit for listeners.

A study by Futrell, Mahowald and Gibson<sup>535</sup> investigated dependency length (the distance between words, in a sentence, that depend on each other; see Figure 7.7) in the use of 37 human languages. The results suggest that speakers attempt to minimise dependency length.

Sentence complexity<sup>286</sup> has a variety of effects on human performance. A study by Kintsch and Keenan<sup>849</sup> asked subjects to read single sentences, each containing the same number of words, but varying in the number of propositions they contained (see Figure 7.8). The time taken to read each sentence and recall it (immediately after reading it) was measured.

The results found that reading rate decreases, as the number of propositions in a sentence increases (with the total number of words remaining the same). Figure 7.9 shows reading time (in seconds) for: 1) sentences containing a given numbers of propositions, and 2) when a given number of propositions were recalled by subjects; with fitted regression models. Some propositions were not recalled, making the model based on recall a more accurate fit to the number of propositions that were processed. A later study<sup>850</sup> found that reader performance was also affected by the number of word concepts in the sentence, and the grammatical form of the propositions (subordinate or superordinate).

### 7.2.3 Memory for material read

What do people remember about the material they have recently read?

Again, the only detailed experimental data available is from studies based on human language prose.

Studies<sup>201</sup> have found that in the short term syntax (i.e., words) is remembered, while over the longer term mostly semantics (i.e., the meaning) is remembered; explicit verbatim memory for text does occur.<sup>644</sup> Readers might like to try the following test (based on Jenkins<sup>773</sup>); part 1: A line at a time, 1) read the sentence on the left, 2) look away and count to five, 3) answer the question on the right, and 4) repeat process for the next line.

The girl broke the window on the porch.

The hill was steep.

The cat, running from the barking dog, jumped on the table.

The tree was tall.

The old car climbed the hill.

The cat running from the dog jumped on the table.

The girl who lives next door broke the window on the porch.

The car pulled the trailer.

The scared cat was running from the barking dog.

The girl lives next door.

The tree shaded the man who was smoking his pipe.

The scared cat jumped on the table.

The girl who lives next door broke the large window.

The man was smoking his pipe.

The old car climbed the steep hill.

The large window was on the porch.

The tall tree was in the front yard.

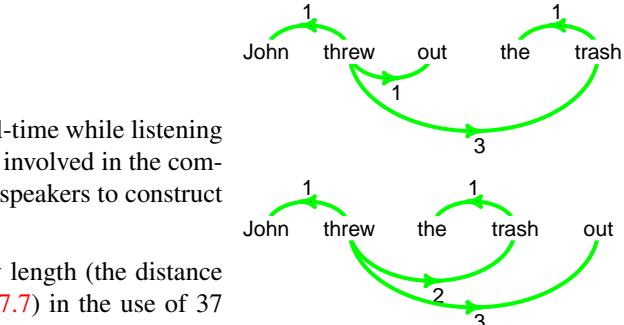


Figure 7.7: Two sentences, with their dependency representations; upper sentence has total dependency length six, while in the lower sentence it is seven. Based on Futrell et al.<sup>535</sup> code

Romulus, the legendary founder of Rome, took the women of the Sabine by force.



Cleopatra's downfall lay in her foolish trust in the fickle political figures of the Roman world.

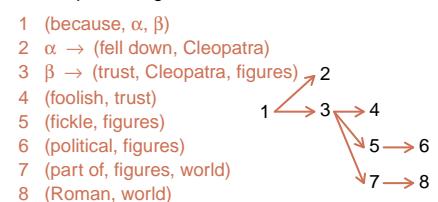


Figure 7.8: One sentence containing four, and the other eight propositions, along with their propositional analyses. Based on Kintsch et al.<sup>849</sup> code

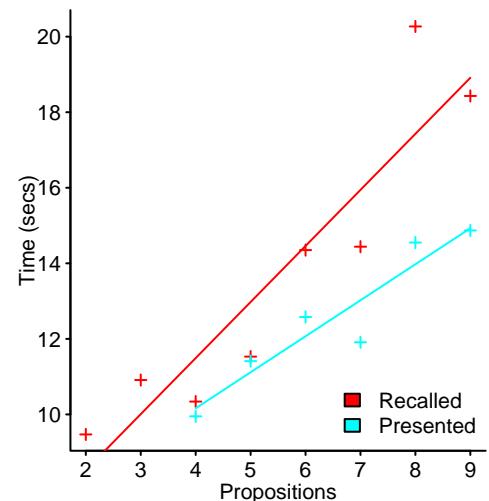


Figure 7.9: Mean reading time (in seconds) for sentences containing a given number of propositions and as a function of the number of propositions recalled by subjects; with fitted regression models. Data extracted from Kintsch et al.<sup>849</sup> code

The car pulling the trailer climbed the steep hill.	Did what?
The cat jumped on the table.	Where?
The tall tree in the front yard shaded the man.	Did what?
The car pulling the trailer climbed the hill.	Which car?
The dog was barking.	Was what?
The window was large.	What was?

You have now completed part 1. Please do something else for a minute, or so, before moving on to part 2 (which follows immediately below).

Part 2: when performing this part, do not look at the sentences above, from part 1. Now, a line at a time, 1) read the sentence on the left, 2) if you think that sentence appeared as a sentence in part 1 express your confidence level by writing a number between one and five (one expressing very little confidence and five expressing a lot of confidence in the decision) next to **old**, otherwise write a number representing your confidence level next to **new**, and 3) repeat process for the next line.

The car climbed the hill.	old____, new ____
The girl who lives next door broke the window.	old____, new ____
The old man who was smoking his pipe climbed the steep hill.	old____, new ____
The tree was in the front yard.	old____, new ____
The window was on the porch.	old____, new ____
The barking dog jumped on the old car in the front yard.	old____, new ____
The cat was running from the dog.	old____, new ____
The old car pulled the trailer.	old____, new ____
The tall tree in the front yard shaded the old car.	old____, new ____
The scared cat was running from the dog.	old____, new ____
The old car, pulling the trailer, climbed the hill.	old____, new ____
The girl who lives next door broke the large window on the porch.	old____, new ____
The tall tree shaded the man.	old____, new ____
The cat was running from the barking dog.	old____, new ____
The cat was old.	old____, new ____
The girl broke the large window.	old____, new ____
The car climbed the steep hill.	old____, new ____
The man who lives next door broke the window.	old____, new ____
The cat was scared.	old____, new ____

You have now completed part 2. Count the number of sentences you judged to be **old**.

The ~~subjective~~ is that all the sentences are new

What is thought to happen is that while reading, people abstract and remember the general ideas contained in sentences (in this case, they are based on the four *idea sets*: 1) “The scared cat running from the barking dog jumped on the table.”, 2) “The old car pulling the trailer climbed the steep hill.”, 3) “The tall tree in the front yard shaded the man who was smoking his pipe.”, and 4) “The girl who lives next door broke the large window on the porch.”).

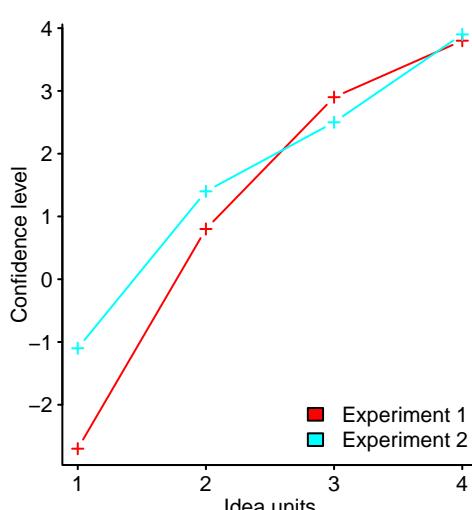
A study by Bransford and Franks<sup>201</sup> investigated subjects' confidence of having previously seen a sentence. Sentences contained either one idea unit (e.g., “The cat was scared.”), two idea units (e.g., “The scared cat jumped on the table.”), three idea units (e.g., “The scared cat was running from the dog.”), and four idea units (e.g., “The scared cat running from the barking dog jumped on the table.”). Subjects saw 24 sentences, after a 4-5 minute break they were shown 28 sentences (24 of which were new sentences), and asked to rank their confidence of having previously seen the sentence (on a 1 to 5 scale).

Figure 7.10 shows that subject's confidence of having previously seen a sentence increases with the number of idea units it contains. New sentences contained one or more idea units that were contained in previously seen sentences. The results are consistent with subject confidence level being driven by the number of previously seen idea units in a sentence, rather than the presence of new idea units.

People use their experience with the form and structure of often repeated sequences of actions, to organize the longer-term memories they form about them. The following studies illustrate the effect that a person's knowledge of the world can have on their memory for what they have read, particularly with the passage of time, and their performance in interpreting sequences of related facts they are presented with:

- A study by Bower, Black, and Turner<sup>190</sup> gave subjects a number of short stories describing various activities (i.e., scripts), such as visiting the dentist, attending a class lecture, going

Figure 7.10: Subject confidence level, on a one to five scale (yes positive, no negative), of having previously seen a sentence containing a given number of idea units. Data extracted from Bransford et al.<sup>201</sup> code



to a birthday party, etc., to read. Each story contained about 20 actions, such as looking at a dental poster, having teeth X-rayed, etc. After a 20-minute interval, subjects were asked to recall actions contained in the stories.

The results found that around a quarter of recalled actions might be part of the script, but were not included in the written story. Approximately seven percent of recalled actions were not in the story and would not be thought to belong to the script.

A second experiment involved subjects reading a list of actions which, in the real world, would either be expected to occur in a known order or not be expected to have any order (e.g., the order of the floats in a parade). The results showed that, within ordered scripts, actions that occurred at their expected location were recalled 50% of the time, while actions occurring at unexpected locations were recalled 18% of the time at that location. The recall rate for unordered scripts (i.e., the controls) was 30%.

- A study by Graesser, Woll, Kowalski, and Smith<sup>612</sup> read subjects stories representing scripted activities (e.g., eating at a restaurant). The stories contained actions that varied in the degree to which they were typical of the script (e.g., Jack sat down at the table, Jack confirmed his reservation, and Jack put a pen in his pocket).

Memory Test	Typical (30 mins)	Atypical (30 mins)	Typical (1 week)	Atypical (1 week)
Recall (correct)	0.34	0.32	0.21	0.04
Recall (incorrect)	0.17	0.00	0.15	0.00
Recognition (correct)	0.79	0.79	0.80	0.60
Recognition (incorrect)	0.59	0.11	0.69	0.26

Table 7.3: Probability of subjects recalling or recognizing typical or atypical actions present in stories read to them, at two time intervals (30 minutes and 1 week) after hearing them. Based on Graesser et al.<sup>612</sup>

The results showed (see Table 7.3) that recall was not affected by typicality over short periods of time, but that after one week recall of atypical actions dropped significantly. Recognition performance for typical vs. atypical actions was less affected by the passage of time (e.g., subjects were asked if a particular action occurred in the story).

- A study by Dooling and Christiaansen<sup>411</sup> asked subjects to read a short biography containing 10 sentences. The only difference between the biographies read by subjects was that in some cases the name of the character was fictitious (i.e., a made up name), while in other cases it was the name of an applicable famous person. For instance, one biography described a ruthless dictator and used either the name Gerald Martin or Adolph Hitler.

After 2-days, and then after 1-week, subjects were given a list of 14 sentences (seven sentences that were included in the biography they had previously read and seven that were not included), and asked to specify which sentences they had previously read.

To measure the impact of subjects' knowledge about the famous person, on recognition performance, some subjects were given additional information. In both cases the additional information was given to the subjects who had read the biography containing the made up name (e.g., Gerald Martin). The *before* subjects were told just before reading the biography that it was actually a description of a famous person and given that persons name (e.g., Adolph Hitler). The *after* subjects were told just before performing the recognition test (they were given one minute to think about what they had been told) that the biography was actually a description of a famous person and given that persons name.

The results (see Figure 7.11) are consistent with the idea that remembering is constructive. After a week subjects memory for specific information in the passage was lost and under these conditions, recognition of sentences is guided by subjects' general knowledge. Variations in the time between reading the biography and identity of a famous character being revealed affected the extent to which subjects integrated this information.

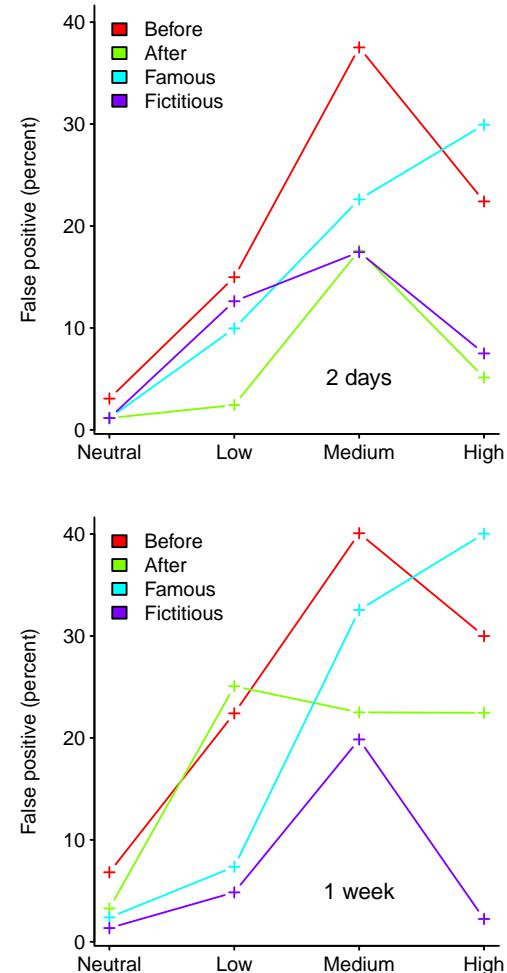


Figure 7.11: Percentage of false-positive recognition errors for biographies having varying degrees of thematic relatedness to the famous person, in *before*, *after*, *famous*, and *fictitious* groups. Data extracted from Dooling et al.<sup>411</sup> code

## 7.2.4 Integrating information

Units of source code (e.g., statements and functions) are combined in a way that results in a behavioral narrative emerging during program execution. Modify an existing behavioral narrative requires that the developer acquire a good enough understanding of how execution

of the units of code combine to cause the behavior to emerge. Developers have to extract information from the units of code and integrate it into a mental model.

What are the factors that have a significant impact the cognitive effort required to integrate source code information into a mental model? Studies<sup>848, 1050</sup> of prose comprehension provide some clues.

The process of integrating two related items of information involves acquiring the first item, keeping it available for recall while processing other information until the second item is encountered; it is now possible to notice that the two items are related, and act on this observation.

A study by Daneman and Carpenter<sup>353</sup> investigated the connection between various measures of subjects' working memory span, and their performance on a reading comprehension task. The two measures of working memory used were the *reading span* and *word span*. In the reading span test subjects have to read, out loud, sequences of sentences while remembering the last word of each sentence, which have to be recalled at the end of the sequence. The number of sentences in each sequence is increased until subjects are unable to successfully recall all the last words.

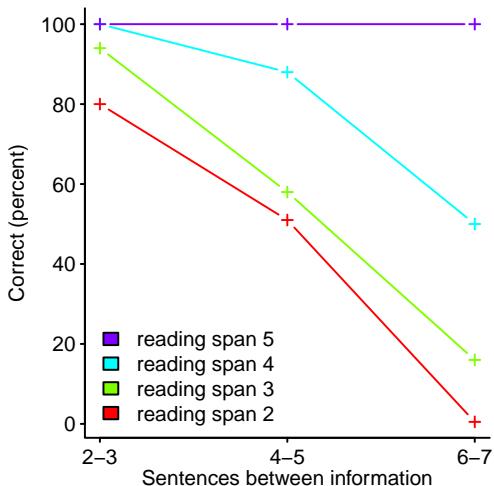


Figure 7.12: Percentage of correct responses, for subjects having a given reading span, to the pronoun reference questions as a function of the number of sentences (x-axis) between the pronoun and the referent noun. Data extracted from Daneman et al.<sup>353</sup> [code](#)

The comprehension task involved subjects reading a narrative passage containing approximately 140 words and then answering questions about facts and events described in the passage. Various passages were constructed such that the distance between the information needed to answer questions varied. For instance, the final sentence of the passage contained a pronoun (e.g., she, her, he, him, or it) that referred to a noun occurring in a previous sentence; different passages contained the referenced noun in either the second, third, fourth, fifth, sixth, or seventh sentence before the last sentence.

In the excerpt: “ . . . river clearing . . . The proceedings were delayed because the leopard had not shown up yet. There was much speculation as to the reason for this midnight alarm. Finally, he arrived, and the meeting could commence.” the question “Who finally arrived?” refers to information contained in the last and third to last sentence. The question “Where was the meeting held?” requires the recall of a fact.

The results showed a strong correlation between their performance in the reading span test, and the reading comprehension test (see Figure 7.12). A similar pattern of results was obtained when the task involved listening, rather than reading. A study by Turner and Engle<sup>1573</sup> found that having subjects verify simple arithmetic identities, rather than a reading comprehension test, did not alter the results. However, altering the difficulty of the background task (e.g., using sentences that required more effort to comprehend) reduced performance.

As a coding example, given the following three assignments, would moving the assignment to `x` after the assignment to `y` reduce the cognitive effort needed to comprehend the value of the expression assigned to `z`?

```
x = ex_1 + ex_2;          /* Could be moved to after assignment to y. */
y = complicated_expression; /* No dependencies on previous statement. */
z = y + ex_1;
```

This question assumes that `ex_2` does not appear prior to the assignment to `x`, in which case there may be a greater benefit to this assignment appearing close to this usage, rather than close to the assignment to `z`.

Is reader cognitive effort reduced by having a single complex statement, rather than several simpler statements?

A study by Daneman and Carpenter<sup>354</sup> investigated subjects performance integrating information within a single sentence, compared to across two sentences, e.g., “There is a sewer near our home who makes terrific suits” (what is known as a *garden path* sentence) and “There is a sewer near our home. He makes terrific suits.” The results found that a sentence boundary can affect comprehension performance. It was proposed that this performance difference was caused by readers purging any verbatim information they held in working memory, about a sentence, on reaching its end. The availability of previously read words, in the single sentence case, making it easier to change an interpretation, based of what has already been read.

A parallel might be drawn between paragraphs in prose and function definitions in code.

A study by Kintsch, Mandel, and Kozminsky<sup>851</sup> investigated the time taken to read and summarize 1,400 word stories. The order of the paragraphs (not the sentences) in the text seen

by some students was randomized. The results showed that while it was not possible to distinguish between the summaries produced by subjects reading ordered vs. randomised stories, reading time for randomly ordered paragraphs was significantly longer (9.05 minutes vs. 7.34).

slicing studies... data?...

The difference between word span and reading span as measures of working memory, is that the first is purely a measure of memory usage while the second also involves processing information (the extent to which processing information consumes, interferes with, or competes with working memory resources is a hotly debated issue). Comprehension of prose involves integrating information within and between sentences. Other studies have found that reading span correlates with performance in a number other tasks, including:

- Reading of prose involves a number of structural components, e.g., at the microstructure level words are formed from character sequences and syntactic processing of words occurs, while at the macrostructure level information has to be integrated across sentences for the narrative to be followed. A study by Graesser, Hoffman, and Clark<sup>611</sup> found that: more cognitive resources are allocated to macrostructure than microstructure processing, differences in reading speed could be attributed to different rates of microstructure processing, and variations in reader goals affect the rate at which the macrostructure is processed.
- A study by Gibson and Thomas<sup>565</sup> found that subjects were likely to perceive complex ungrammatical sentences as being grammatical. Subjects handled complex sentence that exceeded working memory capacity by forgetting parts of the syntactic structure of the sentence, to create a grammatically correct sentence.
- A study by Ehrlich and Johnson-Laird<sup>434</sup> asked subjects to draw diagrams depicting the spatial layout of everyday items specified by a sequence of sentences. The sentences varied in the extent to which an item appearing as the object (or subject, or not at all) in one sentence appeared as the subject (or object, or not at all) in the immediately following sentence. For instance, there is referential continuity in the sentence sequence “The knife is in front of the pot. The pot is on the left of the glass. The glass is behind the dish.”, but not in the sequence “The knife is in front of the pot. The glass is behind the dish. The pot is on the left of the glass.”

The results found that when the items in the sentence sequences had referential continuity 57% of the diagrams were correct, compared to 33% when there was no continuity. Most of the errors for the non-continuity sentences were items missing from the diagram drawn and subjects reported finding it difficult to remember the items as well as the relation between them.

use longer version, eRm package... ?, ?, ?

## 7.2.5 Visual organization

Source code can be visually organised in a wide variety of ways. Visual attention is a limited resource (as illustrated by Figure 7.13) and source with the appropriate layout characteristics can take advantage of the in-built hardware available to the human visual system, to reduce the conscious effort needed to visually process code, e.g., indentation takes advantage of preattentive detection of lines; Section 2.3 discussed human visual processing.

People's ability to learn means that, with practice, they can adapt to handle a wide variety of disparate visual organizations. This learning ability means that given enough practice, developers are likely to be able to learn to rapidly process many code layout rules. Using a visual organization which is already familiar to developers, reduces the start-up cost of learning to use a code layout, i.e., they can make use of prior learning to start closer to their long-term performance.

For instance, how quickly can people achieve a reading performance comparable to that achieved with a familiar layout (based on reading and error rate).

A study by Kolers and Perkins<sup>875</sup> investigated the extent to which practice improved reading performance. Subjects were asked to read pages of text written in various ways; pages contained, normal, reversed, inverted, or mirrored text, as in the following:

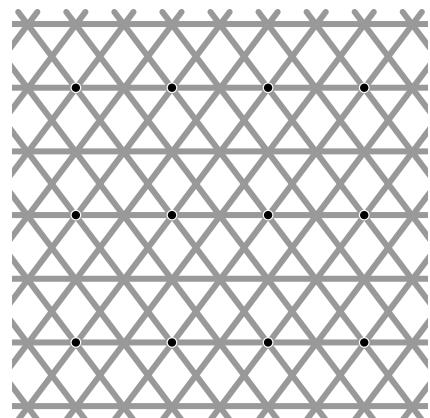


Figure 7.13: Hermann grid, with variation due to Ninio and Stevens<sup>1160</sup> to create an extinction illusion. [code](#)

- Expectations can also mislead us; the unexpected is always hard to perceive clearly. Sometimes we fail to recognize an object because we...

\* .ekam ot detcepxe eb thgm natiruP dnalignE weN a ekatsim fo dnik eht saw tI .eb ot serad eh sa yzal sa si nam yreve taht dias ecno nosremE

- mental processes: Many other reasons can be  
There are but a few of the reasons for believing that a person cannot be conscious of all the information available to them.
- skill or bad visibility of text by people with reading difficulties  
Several years later with readers by psychologists a good measure of intelligence is how many words they can read in one minute.

The time taken for subjects to read a page of text in a particular orientation was measured; the more pages subjects read, the faster they became. In a study<sup>874</sup> a year later, Kolers measured the performance of the same subjects, as they read more pages. Performance improved with practice, but this time the subjects had experience, and their performance started out better and improved more quickly; these results are similar to those obtained in the letter-detection task. Figure 7.14 shows the time taken to read a page containing text in a particular orientation.

## 7.2.6 Identifier names

Identifier names provide a channel for communicating information to readers, that may reduce the effort they need to invest in understanding the source. The communications channel operates through the semantic associations triggered in the mind of the reader of the source. Readers of the source might be developers (including the original author) or tools attempting to gather and use semantic information.

The semantic associations triggered by an identifier may be traced back to information contained in the source (e.g., the declared type of an identifier), or preexisting cultural information present in developers' memory (e.g., meanings associated with words in their native language).

Human languages change over time,<sup>903</sup> with different changes sometimes occurring in different geographic communities. The names used by a community to denote entities evolves over time. One study<sup>355</sup> found a two-stage lifecycle: a linguistically innovative learning phase in which users align with the language of the community, followed by a conservative phase in which users stop responding to changes in community norms...

Source code often contains many different identifiers, each competing for developer cognitive resources. Identifiers having similar spellings, pronunciations, or semantic associations may generate confusion, resulting in mistakes being made; identifiers with long names may consume cognitive resources that are out of proportion to the benefits of the information they communicate. Figure 7.15 shows the number of function definitions containing a given number of occurrences of identifiers, and of distinct identifiers.

Identifiers do not appear in isolation, they occur within a context in source; one study<sup>787</sup> found that identifier names has a large impact on decisions about the precedence of binary operators in expressions.

Desirable characteristics of an identifier name include: triggering the appropriate semantic associations in a readers' mind, a low probability of being mistaken for another identifier present in the associated source code and not consuming a disproportionate amount of cognitive resources.

Identifiers are the most common source token (29% of the visible tokens in the .c files,<sup>785</sup> with comma the second most common at 9.5%), and they represent approximately 40% of all non-white-space characters in the visible source (comments representing 31% of the characters in the .c files).

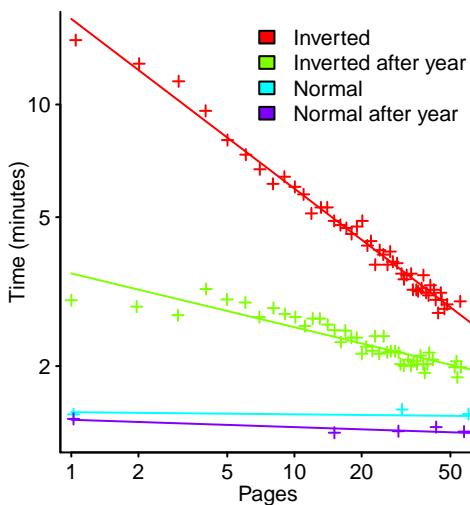


Figure 7.14: Time taken by subjects to read a page of text, printed with a particular orientation, as they read more pages (initial experiment and repeated after one year); with fitted regression lines. Results are for the same six subjects in two tests more than a year apart. Based on Kolers.<sup>874</sup> code

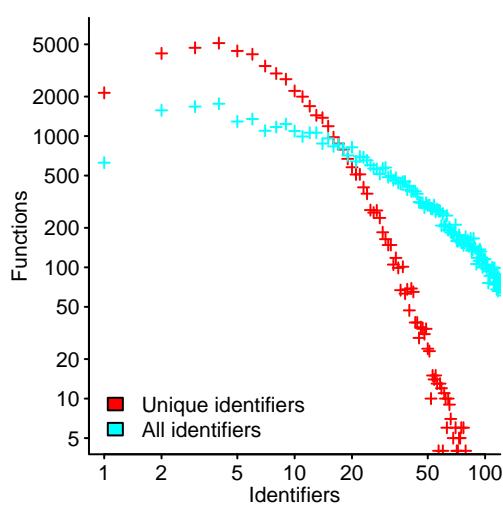


Figure 7.15: Number of C function definitions containing a given number of identifier uses (unique and all). Data from Jones.<sup>785</sup> code

# < . >	include string.h define MAX_CNUM_LEN define VALID_CNUM define INVALID_CNUM	#include <string.h> #define v1 13 #define v2 0 #define v3 1
# 13 # 0 # 1 ( * [] ) , ; * = ( ); { > ) * = ; { ( = < ; ++) (( [] < '0')    ( [] > '9')) * = ; }	int chk_cnum_valid char cust_num int cnum_len cnum_status VALID_CNUM cnum_len strlen cust_num if cnum_len MAX_CNUM_LEN cnum_status INVALID_CNUM else for i i cnum_len i if cust_num i cust_num i cnum_status INVALID_CNUM	int v4(char v5[]), int *v6) { int v7, v8; *v6=v2; v8=strlen(v5); if (v8 > v1) { *v6=v3; } else { for (v7=0; v7 < v8; v7++) { if ((v5[v7] < '0')    (v5[v7] > '9')) { *v6=v3; } }

Figure 7.16: Three versions of the source of the same program, showing identifiers, non-identifiers and in an anonymous form; illustrating how a reader's existing knowledge of words can provide a significant benefit in comprehending source code. Based on an example from Laitinen.<sup>905</sup>

Characteristics of words, in written form, where studies have found an effect on some aspect of subject performance include: *word length effect*,<sup>1147</sup> age of acquisition<sup>332,1387</sup> (when the word was first learned), frequency of occurrence<sup>76</sup> (e.g., in spoken form and various kinds of written material), articulatory features of the initial phoneme (analysis of a character sequences starts at the beginning, so...)...

Historically, most of the studies have involved English, but a growing number of large scale studies involve other languages.<sup>491</sup> Orthography (the spelling system for the written form of a language) can have an impact on reader performance, English has a deep orthography (i.e., a complex mapping between spelling and sound), while Malay has a shallow orthography (i.e., a one-to-one mapping between spelling and sound; also Spanish) and word length in Malay has been found to be a better predictor for word recognition than word frequency<sup>1687</sup> ...

Developers make use of the characteristics of their native language when creating identifier names, e.g., lexical conventions, syntax,<sup>252</sup> word ordering conventions (adjectives). Developers for whom English is a second language are likely to make mistakes, in the use of English, that are influenced by the characteristics of their native language.<sup>1526</sup>

As always, the cost of any investment a developer is willing to make to reduce the cognitive effort of subsequent readers, must be less than the total benefit obtained by all subsequent readers.

Given that most functions are only ever modified by the one developer (e.g., Figure 7.21), the primary beneficiary of any investment in local identifier names is likely to be the developer who made it.

Given that identifier communication occurs via semantic associations, what is the probability that a particular identifier will trigger the same semantic associations in multiple readers of the code?

A study by Nelson, McEvoy and Schreiber<sup>1142</sup> investigated free association of words; subjects were given a word and asked to reply with the first word that came to mind. More than 6,000 subjects producing over 700,000 responses to 5,018 stimulus words.

What is the probability that the same word will be produced by more than one subject? Figure 7.17 shows the mean probability that a given percentage of subjects will produce the same word (from the same cue word). The mean percentage of unique responses was 18% (sd 9).

Table 7.4 shows the percentage of identifiers occurring in each pair of seven large software systems.

Useful semantic associations may be triggered by different words, however, the same word may trigger different semantic associations in different people.<sup>?</sup>

Studies have found that people differ in the meaning they assign to the same word. For instance, the extent to which a word is thought to denote a concrete or abstract concept<sup>1242</sup> (*concrete* words, defined as things or actions in reality, experienced directly through the senses,

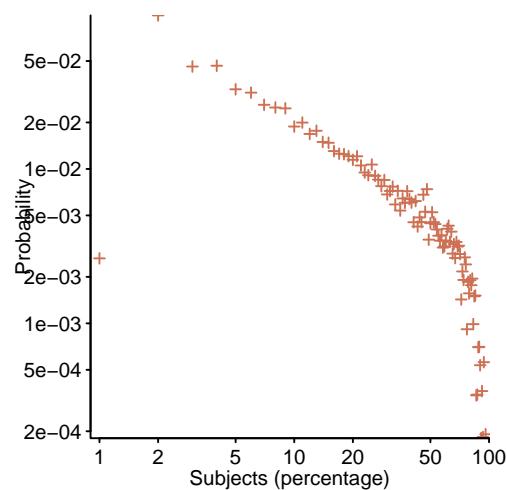


Figure 7.17: Probability (averaged over all cue words) that, for a given cue word, a given percentage of subjects will produce the same word. Data from Nelson et al.<sup>1142</sup> code

whereas *abstract* words are not experienced through the senses, they are language-based with their meaning depending on other words).

	<b>gcc</b>	<b>idsoftware</b>	<b>linux</b>	<b>netscape</b>	<b>openafs</b>	<b>openMotif</b>	<b>postgresql</b>
<b>identifiers</b>	46,549	27,467	275,566	52,326	35,868	35,465	18,131
<b>gcc</b>	-	2	9	6	5	3	3
<b>idsoftware</b>	5	-	8	6	5	4	3
<b>linux</b>	1	0	-	1	1	0	0
<b>netscape</b>	5	3	8	-	5	7	3
<b>openafs</b>	6	4	12	8	-	3	5
<b>openMotif</b>	4	3	6	11	3	-	3
<b>postgresql</b>	9	5	12	11	10	6	-

Table 7.4: Percentage of identifiers in one program having the same spelling as identifiers occurring in various other programs. First row is the total number of identifiers in the program, and the value used to divide the number of shared identifiers in that column. Data from Jones.<sup>785</sup>

A study<sup>1413</sup> of word choice in a fill-in-the-blank task, e.g., “in tracking the . . .”, found that probability of word occurrence (derived from large samples of language use) was a good predictor of the words chosen and the order in which subjects produced them (subjects were asked to provide 20 responses per phrase).

Readers of different natural languages will have trained on different inputs, and during school studied different subjects (each having its own technical terms). A study by Gardner, Rothkopf, Lapan, and Lafferty<sup>545</sup> asked subjects (10 engineering, 10 nursing, and 10 law students) to indicate whether a letter sequence was a word or a nonword. The words were drawn from a sample of high frequency words (more than 100 per million), medium-frequency (10–99 per million), low-frequency (less than 10 per million), and occupationally related engineering or medical words.

The results showed engineering subjects could more quickly and accurately identify the words related to engineering (but not medicine); the nursing subjects could more quickly and accurately identify the words related to medicine (but not engineering). The law students showed no response differences for either group of occupationally related words. There were no response differences on identifying nonwords. The performance of the engineering and nursing students on their respective occupational words was almost as good as their performance on the medium-frequency words.

Dearest creature in creation,  
Study English pronunciation.  
I will teach you in my verse

Sounds like corpse, corps, horse, and worse.  
I will keep you, Suzy, busy,  
Make your head with heat grow dizzy.  
Tear in eye, your dress will tear.  
So shall I! Oh hear my prayer.  
Pray, console your loving poet,  
Make my coat look new, dear, sew it!

Just compare heart, beard, and heard,  
Dies and diet, lord and word,  
Sword and sward, retain and Britain.  
(Mind the latter, how it's written.)  
Now I surely will not plague you  
With such words as plaque and ague.  
But be careful how you speak:

Say break and steak, but bleak and streak;  
Cloven, oven, how and low,  
Script, receipt, show, poem, and toe.

THE CHAOS (first two verses)  
by Dr. Gerard Nolst Trenité, 1870-1946

What characteristics increase the likelihood that an identifier will be mistaken for another one?

A study by Lambert, Chang, and Gupta<sup>907</sup> investigated drug name confusion errors.<sup>v</sup> Subjects briefly saw the degraded image of a drug name. Both the frequency of occurrence of drug names, and their neighborhood density were found to be significant factors in the error rate.

An analysis of the kinds of errors made, found that 234 were omission errors and 4,128 were substitution errors. In the case of the substitution errors, 63.5% were names of other drugs (e.g., **Indocin®** instead of **Indomed®**), with the remaining substitution errors being spelling-related or other non-drug responses (e.g., **Catapress** instead of **Catapres®**).

A study by Kernighan, Church, and Gale<sup>836</sup> investigated patterns of incorrect spellings; 14,742 *misspelled* words detected from 44 million words of text were used to build four tables of probabilities. For instance, of times the character *y* is deleted when it follows the character *x* (in misspelled words) divided by the number of times the character *y* follows the character *x* (in all words in the text). The other tables counted insertions, substitutions, and transpositions...

Pronunciation can be context dependent, e.g., “You can lead a horse to water, but a pencil must be lead.” and “Wind the clock when the wind blows.”

Identifiers often contain character sequences that do not match words in the native language of the reader. Studies of prose have included the use non-words, often as a performance comparison against words. Nonwords are sometimes read as a word whose spelling it closely resembles.<sup>1272</sup>

Studies of letter similarity have a long history,<sup>1556</sup> and tables of visual<sup>1112</sup> (e.g., 1 (one) and 1 (ell)) and acoustic<sup>1243</sup> letter confusion have been published. When categorizing a stimulus,

<sup>v</sup> Errors involving medication kill one person every day in the U.S., and injure more than one million every year; confusion between drug names that look and sound alike account for 15% to 25% of reported medication errors

people are more likely to ignore a feature than they are to add a missing feature, e.g., **Q** is confused with **O** more often than **O** is confused with **Q**.

A study by Avidan<sup>74</sup> asked subjects to work out what a method did, timing how long it took them to reply. In the control condition subjects saw the original method and in the experimental condition the method name was replaced by xxx and local and parameter names were replaced by single letter identifiers; in all cases the method name was replaced by xxx.

Subjects took longer to reply for modified methods. When parameters were revealed first, subjects were 268 seconds slower (on average), and when locals were revealed first 342 seconds slower (the standard deviation of the between subject differences was 187 and 253 seconds, respectively; see [sourcecode/Avidan-MSc.R](#)).

The names of existing identifiers are sometimes changed.<sup>56</sup> The constraints on identifier renaming include the cost of making all the necessary changes in the code and dependencies other software may have on existing names (e.g., identifier is in a published API).

A study<sup>257</sup> of the effectiveness of two code obfuscation techniques (renaming identifiers and complicating the control flow) found that renaming identifiers had a larger impact on the time taken by subjects to comprehend and change code (see [sourcecode/AssessEffectCodeOb.R](#)).

One study<sup>711</sup> found that the time taken to find a mistake in a short snippet of code was slightly less when the identifiers were words (see [sourcecode/shorter-iden.R](#)).

## 7.2.7 Consistency

People can subconsciously learn and make use of patterns in events that occur in their environment (see Section 2.5). Consistently following a set of patterns, when writing source code, creates an opportunity for readers to make use of this ability. Individual developers do write code in a distinct style,<sup>242</sup> even if they cannot articulate every pattern of behavior they follow.

A study by Lewicki, Hill and Bizot<sup>948</sup> demonstrates the impact of implicit learning on subjects' performance, on a task that contains no overt learning component. While subjects watched a computer screen, a letter was presented in one of four possible locations; subjects had to press the button corresponding to the location of the letter as quickly as possible. The sequence of locations used followed a consistent, but complex, pattern. The results showed subjects' response times continually improving as they gained experience. The presentation was divided into 17 segments of 240 trials (a total of 4,080 letters). The pattern used to select the sequence of locations was changed after the 15th segment, but subjects were not told about the existence of any patterns of behavior. After completing the presentation subjects were interviewed to find out if they had been aware of any patterns in the presentation; they had not.

Figure 7.18 shows the mean response time for each segment. The consistent improvement, after the first segment, is interrupted by a decrease in performance after the pattern changes.

Implicit learning of grammars...

A study by Corazza, Maggio and Scanniello<sup>327</sup> investigated what they called the *coherence* between a method's implementation and any associated comment (i.e., does the method implement the intent expressed in the comment). Five Java programs (5,762 methods) were manually evaluated; coherence was positively correlated with  $\log(\text{comment\_lines})$  and negatively correlated with method LOC (see [sourcecode/SQJ\\_2015.R](#)).

A study by Buse and Weimer<sup>234</sup> investigated Computer Science students' opinions on the readability of short snippets of Java source code, rating them on a scale of 1 to 5. The students were taking first, second and third/fourth year Computer Science degree courses and postgraduates at the researchers' University.

Subjects were not given any instructions on how to rate the snippets for readability, and the outcome that individuals were trying to achieve when rating is not known, e.g., were subjects rating based on how readable they personally found the snippets to be, or rating on the basis of the answer they would expect to give if they were being tested in an exam. The study can be viewed as an investigation of implicit learning, with between student consistency of readability ratings improving as students progressed through the years of attending computer science courses.

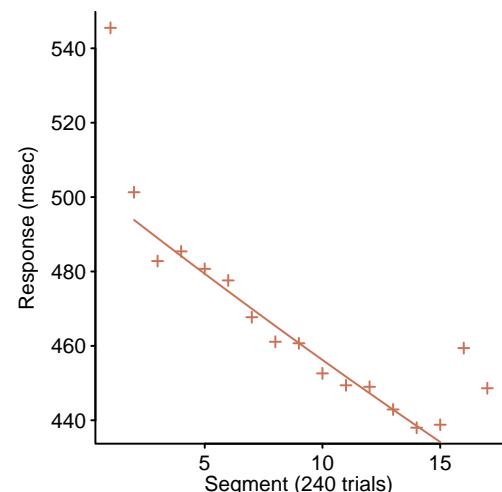


Figure 7.18: Mean response time for each of 17 segments; regression line fitted to segments 2-15. Data extracted from Lewicki et al.<sup>948</sup> code

The results show that the agreement between students readability ratings, for short snippets of code, improves as students progress through course years 1 to 4 of a computer science degree (see [developers/readability](#)).

?

Readability as a categorization problem...

### 7.2.8 Folklore metrics

Two source code metrics, proposed in the 1970s, have become established folklore of software development: Halstead's metric and McCabe's cyclomatic complexity metric. Use of these metrics persists, despite the studies showing them performing little better than counting lines of code. Why do developers and researchers continue to use them? Perhaps counting lines of code is considered to lack the mystique that a young field craves, or it is simply that developers are researchers have not been held accountable for inaccurate predictions.

Ptolemy's theory (i.e., the Sun and planets revolved around the Earth) was not discredited because it gave incorrect answers, but because Copernicus created a much simpler theory that produced results that eventually gave a better fit to experimental measurements (once observing equipment became more accurate).

The studies investigating what became known as Halstead's complexity metric are documented in a series of technical reports<sup>[229, 529, 600, 651, 652, 1198](#)</sup> published in the 1970s. The later reports compared theory with practice, but used very small datasets; Halstead's early work<sup>[652](#)</sup> is based on experimental data obtained for other purposes,<sup>[1716](#)</sup> and later work<sup>[529](#)</sup> used a data set containing nine measurements of four attributes (see [faults/halstead-akiyama.R](#)), others contained 11 measurements<sup>[600](#)</sup> or used measurements from nine modules.<sup>[1198](#)</sup>

Figure 7.19 shows the strong correlation between lines of code, Halstead's volume and cyclomatic complexity...

Researchers of the period found that Halstead's metric did not stand up to scrutiny: Magidin and Viso<sup>[1007](#)</sup> used larger datasets (50 randomly selected algorithms...) and found many faults in the methodology in Halstead's papers; a report by Shen, Conte and Dunsmore,<sup>[1423](#)</sup> Halstead's colleagues at Purdue, written four years after Halstead's flurry of papers, contains a lot of background material and points out the lack of any theoretical foundation for some of the equations, that the analysis of the data was weak and that a more thorough analysis suggests theory and data don't agree.

An observation in the Shen et al report, that Halstead originally proposed the metrics as a way of measuring the complexity of algorithms not programs, explains the background to Halstead's uses of the term *impurities*, and the discussion for the need for *purification* in his early work. Halstead points out<sup>[229](#)</sup> that the value of metrics for *algorithms written by students* are very different from those for the equivalent programs published in journals and goes on to list eight classes of impurity that need to be purified (i.e., removing or rewriting clumsy, inefficient or duplicate code) in order to obtain results that agree with the theory (i.e., cleaning data in order to obtain results that more closely agree with theory).

The paper in which McCabe<sup>[1034](#)</sup> discussed what he called *complexity measures* contained a theoretical analysis of various graph-theoretic complexity measures, and some wishful thinking that these might apply to software; no empirical evidence was presented. Studies<sup>[91](#)</sup> have found a strong correlation between lines of code and McCabe's complexity metric. This metric also suffers from the problem of being very easy to manipulate (see Section 6.4.2).

## 7.3 Patterns of usage

Patterns of source code usage are generated by processes such as developer beliefs and habits, application requirements, and hardware or software limits (e.g., maximum memory limits, support for a language subset).

Speakers of human languages have common patterns of word usage<sup>[937](#)</sup> and phrase usage,<sup>[157](#)</sup> the written form also has common patterns of letter usage.<sup>[793](#)</sup>

?

Figure 7.19: Lines of code, Halstead's volume and cyclomatic complexity of Linux version 2.6.9. Data from Israel et al.<sup>[762](#)</sup> [code](#)

Individual developer patterns of usage exist,<sup>242</sup> which introduces random amounts of bias into population measurements. These *accents* come from experience with using other languages, suggestions copied from coding techniques appearing in books, course notes, etc.

Are there common patterns specific to application domains, programming language or large organizations? A study by Jones<sup>785</sup> analysed desktop computer applications written in C. While several large source code measurements projects have made their data available<sup>7</sup> and Java,<sup>620</sup> there has not been any other extensive analysis of source usage.

Patterns of usage are of interest to the extent they provide information which aids understand of the software development process. Common usage patterns are also of interest to compiler writers wanting to reduce the effort needed to implement code optimizations (i.e., by focusing on commonly occurring constructs), by static analysis tools interested in mistakes commonly made by developers, and by teachers looking to minimise what students have to know to be able to handle most situations.

Error patterns are also of interest... automate the process of locating and correcting common novice programmer mistakes<sup>780</sup>...

Runtime patterns of complete programs<sup>1328</sup> can be used to help tune the performance of both the measured program and any associated runtime system; effort might also be focused on individual language constructs.<sup>1327</sup>

A theory is of no use unless it can be used to make predictions that can be verified (or not), and any theory of developer coding behavior needs to make predictions about detectable patterns in code. For instance, given the hypothesis that developers are more likely to create a separate function for heavily nested code, then the probability of encountering an if-statement should decrease with increasing nesting depth.

Figure 7.20 shows the number of selection-statements, occurring at a given maximum nesting depth, in C code;<sup>785</sup> an if-statement is the most common form of selection-statement. The probability of encountering a selection-statement does not change with depth of nesting, implying that developers are not more likely to create a function for more deeply nested statements.

emailed, promised...?

Tailored Source Code Transformations to Synthesize Computationally Diverse Program Variants<sup>115</sup>

A study by Taylor<sup>1539</sup> investigated author contribution patterns to Eclipse projects. Figure 7.21 shows the number of files modified by a given number of people; the fitted regression model is  $Files = e^{-0.87authors + 0.01authors^2}$ .

The techniques used to find a solution to an application requirement can impact the use of language constructs. For instance, numerically intense calculations (such as scientific or engineering calculations) often involve floating-point values and multi-dimensional arrays, features that are not often seen in programs targeting other kinds of applications, for loops...?

Most files are only ever edited by one person...

Pattern detection tools can add noise, or to be more exact the source code matched by tools does not always have the sought after characteristics. Also, it is not always possible to obtain definitive answers for some questions about source code characteristics, e.g., code similarity when checking for clones; tools often provide options for controlling their performance.<sup>1306</sup>

A variety of developer practices can also introduce noise into source code measurement data (apart from personal coding habits), these include the following:

- When making use of source written by third-parties, it may be more cost effective to maintain a local copy of the source files, than link to the original. A consequence of this behavior is the presence of duplicate source files in public repositories (skewing population measurements) and individual project measurements may be heavily influenced by the behavior of developers on other projects.

A study by Lopes, Maj, Martins, Saini, Yang, Zitny, Sajnani and Vitek<sup>977</sup> investigated duplicate code in 4.5 million non-fork Github hosted projects (i.e., any forks of a project were not included in the analysis). Of the 428+ millions source files written in Java, C++, Python or Javascript, 85 million were unique (percentage unique files by language...) While not

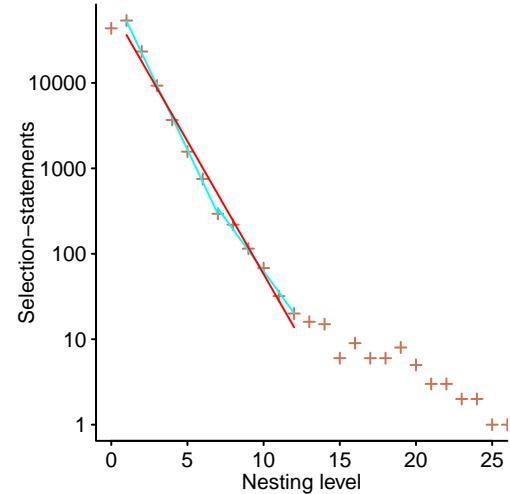


Figure 7.20: Number of selection-statements having a given maximum nesting level; for embedded C<sup>446</sup> (whose data was multiplied by a constant to allow comparison; the data for nesting depth 5 was interpolated from the adjacent points) and based on the visible form of the .c files. Data from Jones.<sup>785</sup> code

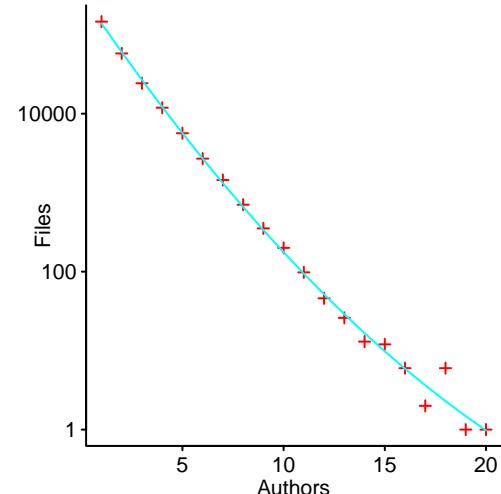


Figure 7.21: Number of files, in Eclipse projects, that have been modified by a given number of people, with fitted regression model. Data from Taylor.<sup>1539</sup> code

	Java	C++	Python	JavaScript
<b>Unique files</b>	60%	27%	31%	7%
<b>Projects</b>	1,481,468	364,155	893,197	1,755,618
<b>duplicates &gt; 50%</b>	14%	25%	18%	48%
<b>duplicates &gt; 80%</b>	9%	16%	11%	31%
<b>duplicates 100%</b>	6%	7%	6%	15%

Table 7.5: Percentage of unique files for a given language, number of projects and average percentage of duplicated files in projects in Github hosted projects written in various languages. Data from Lopes et al.<sup>977</sup>

explicitly labeled as forks, some projects contained a high percentage of files appearing in other projects.

Figure 7.22 shows the number of Python files containing a given number of SLOC, for a 10% sample of all 31,602,780 files and the 9,157,622 unique files.

- When modifying source, some developers commit every change they make (to the project repository), while other developers only make public commits of software they have tested (i.e., they commit changes of behavior, not changes of code).

Figure 7.23 shows the size of commits involved in fixing reported faults in Linux.

also see Figure 8.15...

In many ways existing practice is a *meme machine*.<sup>7</sup> Developers read existing code, learn about the ideas it contains, and potentially use those ideas to write new code. Particular ways of writing code need not be useful to the program that contains them; they only need to appear to be useful to the developer who writes the code, or fit in with a developer's preferred way of doing things.

A study by Kamiya<sup>816</sup> investigated how lines of code that had been deleted, were added back in to the source in a later revision... [sourcecode/iwsc2011-kamiya.R](#)

?

???

### 7.3.1 Language features

The idea that the human language we use influences our thinking is known as the *Sapir-Whorf* or *Whorfian* hypothesis.<sup>557</sup> The *strong language-based* view is that the language used influences its speakers' conceptualization process; the so-called *weak language-based* view is that linguistic influences occur in some cases, such as the following:

- language-as-strategy*: language affects speakers performance by constraining what can be said succinctly with the set of available words; a speed/accuracy trade-off, approximating what needs to be communicated in a brief sentence rather than using a longer sentence to be more accurate,<sup>741</sup>
- thinking-for-speaking*: for instance, English uses count nouns, which need to be modified to account for the number of items, which requires speakers to pay attention to whether one item or more than one item is being discussed; Japanese nouns make use of classifiers, e.g., shape classifiers such as hon (long thin things) and mai (flat things), and measuring classifiers such as yama (a heap of) and hako (a box of). Some languages assign a gender to object names, e.g., the Sun is feminine in German, masculine in Spanish and neuter in Russian.

*thinking for coding* occurs when creating names for identifiers, where plurals may sometimes be used (e.g., the `rowsum` and `rowSums` functions in R),<sup>vi</sup>

- languages vary in the way they articulate numbers containing more than one digit, e.g., the components contained in 24 might be ordered as 20 + 4 (English) or 4 + 20 (French). Studies<sup>222</sup> have investigated linguistic influences on numerical cognition.

<sup>vi</sup> Your author is not aware of any studies investigating the impact of gender on the identifier names used by native speakers of different languages.

While different languages may contain different ways of describing the world, common usage patterns can be found. A study by Berlin and Kay<sup>149</sup> isolated what they called the *basic color terms* of 98 languages. They found that the number and kind of these terms followed a consistent pattern (see Figure 7.24); while the boundaries between color terms varied, the visual appearance of the basic color terms was very similar across languages. Simulations of the evolution of color terms<sup>104</sup> suggest that it takes time for the users of a language to reach consensus on the naming of colors, and over time languages accumulate more color terms.

Languages vary in their complexity (i.e., there is no mechanism that ensures all languages are equally complex<sup>1055</sup>).

Many programming languages in common use are still evolving, i.e., the semantics of some existing constructs are changing and support for new constructs is being added. Changing the semantics of existing constructs involves a trade-off between alienating the developers and companies currently using the language (by failing to continue to process existing code) and fully integrating new constructs into a language.

At the end of 2008 the Python Software Foundation released Python 3, a new version of the language that was not compatible with Python 2. Over time features only available in Python 3 have been back-ported to Python 2. How have Python developers responded to the availability of two similar, but incompatible languages?

A study by Malloy and Power<sup>1012</sup> investigated the extent to which 50 large Python applications were compatible with various releases of the Python language. Figure 7.25 shows changes in the estimated mean compatibility of the 50 applications to 11 versions of Python, over time.

While new features are often added to languages, it is rare for a feature to be removed (at least without giving many years notice). The ISO Standards for both the Fortran and C have designated some constructs as deprecated, i.e., to be removed in the next release...

Whatever the reason for the additions, removals, changes to a language, such changes will influence the characteristics of the code written by developers. A new construct may add new functionality (e.g., atomics in C and C++), displace use of an existing construct (e.g., lambda expressions replacing anonymous classes<sup>1030</sup>)...

How quickly are new language constructs adopted and regularly used by developers? Use of language constructs depends on:

- compiler support. While vendors are quick to claim compliance with the latest standard, independent evidence is rare (e.g., compiler validation by an accredited test lab) footnote:[Your author is not aware of any studies that have investigated the delay between a feature being added to a language standard and vendor support for it; Zortech once claimed 100% C Standard compliance for their compiler (it was not illegal to claim compliance to a non-existent standard and when the standard was published their adverts switched to claiming 99% compliance a meaningless claim)].
- existing developer knowledge and practices. What incentives do existing users of a language have to learn new language features, and to spend time changing existing habits? Is new language feature usage primarily driven by developers new to the language (i.e., learned about the new feature as a by-product of learning the language)?... data...

A handful of compilers now dominate the market for many widely used languages.<sup>vii</sup>

For extensive compiler driven language usage to exist, widely used diverse compilers are required (something that was once common for some languages). The small number of distinct compilers that are now widely used, means that diversity of language usage is likely to be driven by the evolution of compiler support for new language constructs and the extent to which source has been modified to adapt to changed and new features.

Studies of the introduction of generics in C#<sup>842</sup> and Java<sup>1214</sup> found that while developers make use of functionality defined using generics, in libraries, they rarely defined their own generic classes. Also, existing code in most projects is not refactored to use generics, and that the savings (in lines of code) are small.

<sup>vii</sup> The availability of good enough open source compilers has led to most independent compiler vendors leaving the market. A vendor of C validation suites (selling at around \$10,000), once told your author they had over 150 licensees; a non-trivial investment for a compiler vendor.

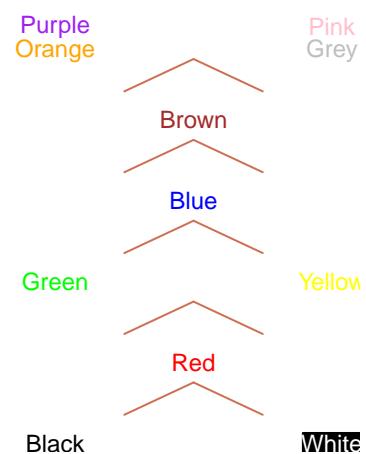


Figure 7.24: The Berlin and Kay<sup>149</sup> language color hierarchy. The presence of any color term in a language implies the existence, in that language, of all terms below it. Papuan Dani has two terms (black and white), while Russian has eleven (Russian may also be an exception in that it has two terms for blue.) code

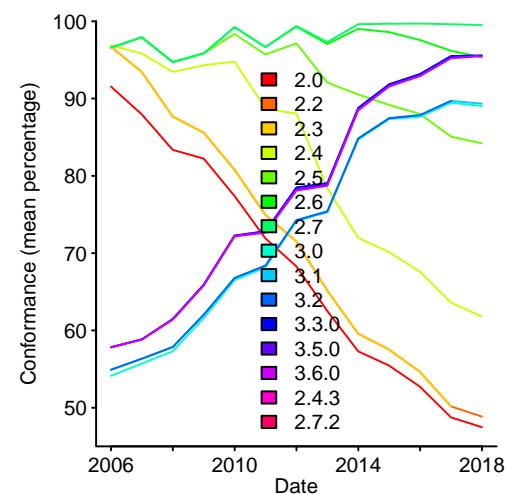


Figure 7.25: Mean compatibility of 50 applications to 11 versions of Python, over time. Data from Malloy et al.<sup>1012</sup> code

Dates on which newly added language features were first used in a file, on a project or by a committer...

A study by Dyer, Rajan, Nguyen and Nguyen<sup>425</sup> investigated the use of newly introduced Java language features, based on the source of commits made to projects on SourceForge. Figure 7.26 shows the cumulative growth (adjusted for the growth of registered SourceForge users<sup>1691</sup>) in the number of developers who had checked in a file containing a usage of a given new feature, for the first time. Possible reasons for the almost complete halt in growth of developers using a new Java language construct for the first time include: the emptying of the pool of developers willing to learn and experiment with new language features, and developers switching to other software hosting sites (Github started in 2008).

### 7.3.2 Runtime characteristics

There is a symbiosis between cpu design and existing code; developers interested in efficiency attempt to write code that makes efficient use of cpu functionality and cpu designers attempt to optimise hardware performance for commonly occurring coding constructs (e.g., locality of reference makes it worthwhile supporting a cache). Hardware characteristics can have a significant impact on performance, e.g., Figure 13.17.

During program execution, patterns in the use of storage by distinct variables include: storage obtained via dynamic allocation usually involves relatively small objects, with program-specific sizes taking up most of the requests,<sup>1018</sup> most dynamically allocated storage has a short lifetime,<sup>1018</sup> and dynamically allocated objects declared using same type name tend to have similar lifetimes.<sup>1448</sup>

A study by Suresh, Swamy, Rohou and Seznec<sup>1523</sup> investigated the value of arguments passed to transcendental functions. Figure 7.28 shows...

Conditional branches have a high degree of predictability cbook?<sup>2</sup>...

When execution time needs to be minimised, the relative performance of semantically equivalent coding constructs are of interest.

A study by Flater and Guthrie<sup>509</sup> compared the time taken to assign a value to an array element in C and C++, using various language constructs. The fitted regression model contained interactions between almost every characteristic measured (see [benchmark/bnds\\_chk.R](#)).

### 7.3.3 Narrative structures

Everyday activities have a structure and include recurring themes. It is to be expected that software systems, operating within these activities, will also have a structure and include recurring themes. A study by Propp<sup>1285</sup> was perhaps the first to study recurring themes, in his analysis of Russian folktales.

A few studies have attempted to find narratives in software, e.g., so called *programming plans*.<sup>1673</sup> Computer games implemented using an Interactive Fiction approach contain source code at a high level of abstraction; the code narrative is essentially the game narrative.<sup>1023</sup>

The functionality provided by a program emerges from the execution specific sequences of code, and this code may be scattered over the complete source used to build a program.

A study by Wong, Gokhale and Horgan<sup>1676</sup> investigated the execution of basic blocks, within the 30 source files making up the SHARPE program, when fed inputs that exercised six of the supported features (in turn). Figure 7.29 shows the fraction of basic blocks in each file executed when processing input exercising a particular feature.

Modularity and information hiding are two commonly used techniques for partitioning the code narratives. The costs and benefits of using this approach are...

When processing source, readers have to assume that everything is relevant, until proven otherwise. Modularity provides a means of reducing the amount of information that needs to be processed by readers of the code... Prior to modifying code, the impact of the changes need to be understood. When uses of a variable are scattered about the source, ...

Localising the impact of mistakes... Reduces the likelihood that a mistake in one part of the code will impact other parts of the code...

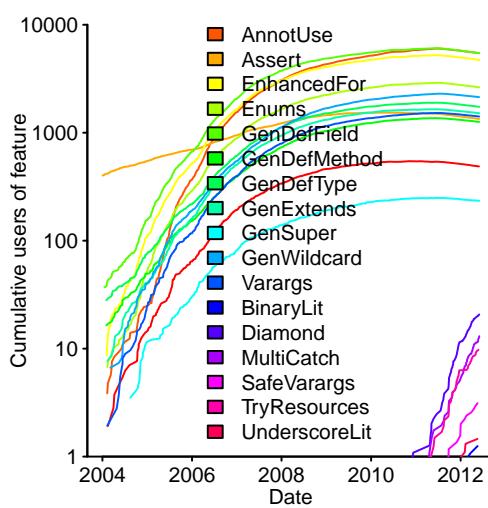


Figure 7.26: Cumulative number of developers who have committed Java source making use of new features added to the language. Data from Dyer et al.<sup>425</sup> [code](#)

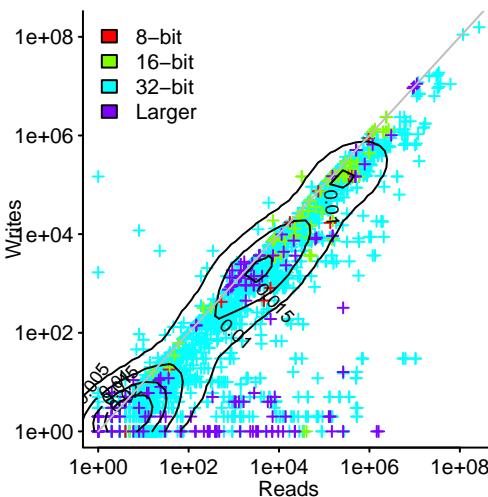


Figure 7.27: Number of reads and writes to the same variable, for 3,315 variables, made during the execution of the Mediabench suite. Data kindly provided by Caspi.<sup>254</sup> [code](#)

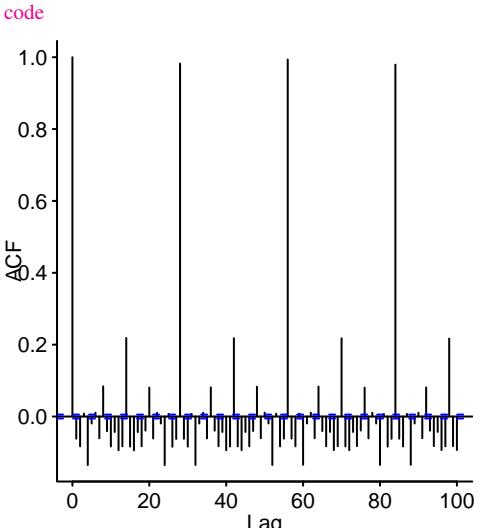


Figure 7.28: to be decided... Data kindly provided by Suresh.<sup>1523</sup> [code](#)

As the amount of functionality supported by a program grows, it may become more profitable to build targeted configurations of the program (i.e., programs supporting a subset of the available features), rather than one program supporting all features. Reasons for supporting multiple configurations include targeting a particular market segment and reducing program resource usage, e.g., the likely cpu time and memory consumed when running the program with the options enabled/disabled.<sup>1438</sup>

Rather than maintaining multiple copies of the source, conditional compilation may be used to select the code that is included in a particular program build. One form of conditional compilation directive tests whether an identifier has been defined as a macro; a program feature is made optional by defining (or not) the corresponding identifier (sometimes known as a *feature test macro*, *feature constant*, or *build flag*). Figure 7.6 shows the number of unique *feature constants* appearing in programs containing a given number of lines of code.

emailed, promised...?

A study by Berger, She, Lotufo, Wąsowski and Czarnecki investigated the interaction between build flags and optional program features in 13 open source projects. Figure 7.30 shows the number of optional features that are enabled when a given number of build flags are set.

How extensive is the impact of build flags on source code? A study by Ziegler, Rothberg and Lohmann<sup>1713</sup> investigated the number of source files in the Linux kernel affected by configuration options. Figure 7.31 shows the number of files affected by the cumulative percentage of configuration options; the impact of 37.5% of options is restricted to a single file, and some options have an impact over tens of thousands of files.

### 7.3.4 Multiple languages

A method's source code may contain multiple languages, e.g., SQL and assembler<sup>1333</sup>...

### 7.3.5 Control flow

The ordering of operations is fundamental to producing the desired program behavior. Control flow statements specify possible orderings of code execution. acquire details about control flow and data flow<sup>1232</sup>

An if-statement represents a decision point, its presence comes from either an application requirement or an internal house-keeping issue (e.g., an algorithmic requirement or checking for an error condition).

Figure 7.32 shows...

Loop statements are where most programs spend most of their time executing (the characteristics of code within loops is intensively studied by compiler writers, optimizing loop code has the greatest cost/benefit).

The **goto** statement was the first form of control flow. The introduction of so called *structured programming*, along with the spread of language support for other forms of control flow statements, saw a significant reduction in its use. The **goto** statement in Ada and C is mostly used to jump out of nested constructs, to statements appearing either earlier or later,<sup>555,785</sup> one study<sup>1127</sup> found that 80% of usage in C was related to error handling.

Exception handling (known as *signal handling* in some languages) often involves control flow between functions/methods. Use of exception handling in C# and Java...?

emailed for data...?

Many languages support a **switch** statement to handle the situation where multiple conditions may be true at some point. In some cases, developers may choose to use a sequence of if-statements, e.g., the following is from the source of a version of grep:

```
if ((c = *sp++) == 0)
    goto cerror;
if (c == '<') { ... }
if (c == '>') { ... }
if (c == '[') { ... }
if (c == ']') { ... }
if (c >= '1' && c <= '9') { ... }
```

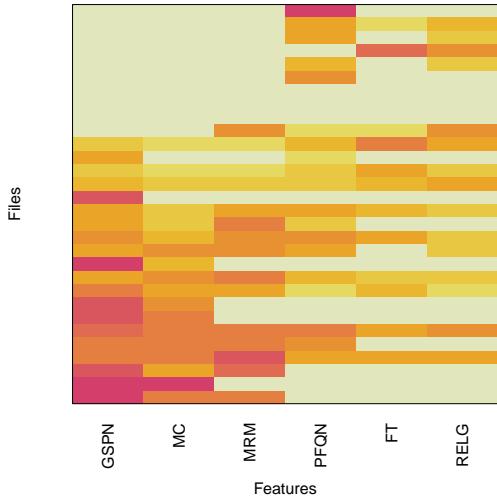


Figure 7.29: Heat map of the fraction of a file's basic blocks executed when performing a given feature of the SHARPE program. Data from Wong et al.<sup>1676</sup> [code](#)

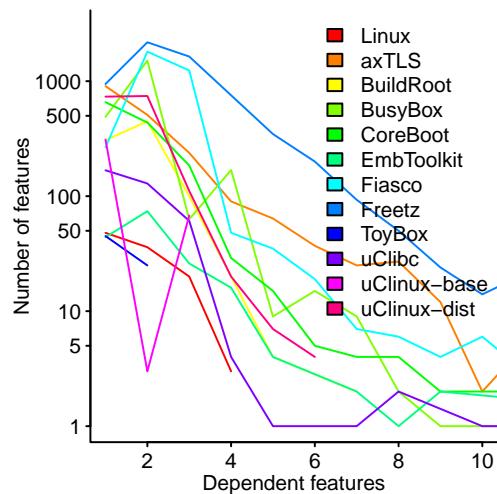


Figure 7.30: Number of optional features selected by a given number of flags. Data kindly provided by Berger.<sup>147</sup> [code](#)

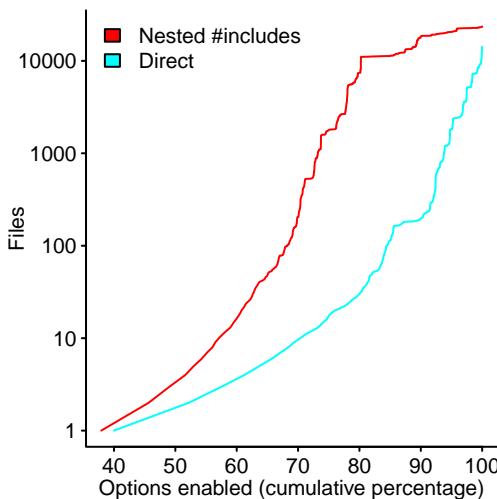


Figure 7.31: Cumulative percentage of configuration options impacting a given number of source files in the Linux kernel. Data kindly provided by Ziegler.<sup>1713</sup> [code](#)

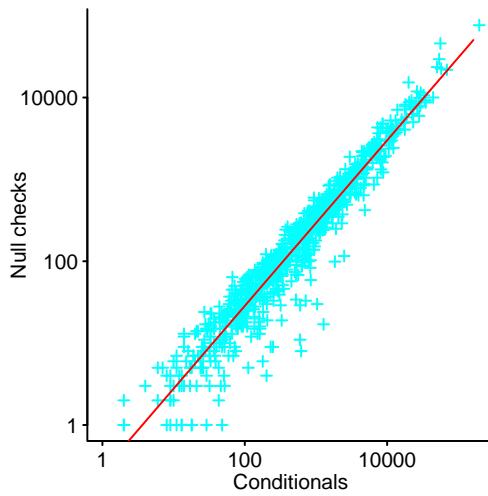


Figure 7.32: Total if-statements against if-statements whose condition involves a null check, in each of 800 Java projects. Data kindly provided by Osman.<sup>1196</sup> [code](#)

The ordering of if-statements may be driven by developer beliefs about the most efficient order to perform the tests, always adding new tests last, or some other reason. Evolution of control flow usage, a study by Jones<sup>788</sup> ...

A study by Yang, Uh, and Whalley<sup>1684</sup> used profile information to reorder if-statements, by frequency of their conditional expression evaluating to true. The average performance improvement, on a variety of Unix tools, was 4%.

Figure 11.30 shows...

Growth in complexity of conditions with SLOC... [sourcecode/1-s2.R](#) and [sourcecode/sast\\_2017.R](#)

### 7.3.6 Statements

Statements have been the focus of the majority of studies of source code... emailed for data...?

Some languages support the creation of executable code at runtime, e.g., concatenating characters to build a sequence corresponding to an executable statement and then calling a function that interprets the string just as-if it appeared in a source file.

A study by Rodrigues and Terra<sup>1343</sup> investigated the use of dynamic features by 28 Ruby programs. On average 2.6% of the language features appearing in the source were dynamic features; it was thought possible to replace 50% of the dynamic statements with static statements (a consequence of developer habit of using template driven solutions...).

Figure 7.33 shows the number of dynamic statements, LOC, and methods appearing in Ruby programs containing a given number of those constructs. Lines are a power law fit, with the exponent varying between 0.8 and 0.9.

#### 7.3.6.1 Literal values

Literal values appear in the source of a variety of reasons, including: required by the algorithm,<sup>57</sup> size or number of elements in declarations of array types,<sup>785</sup> implementations specific values (e.g., urls, dates, etc), application domain values and personal choices of developers (for a discussion of some of the factors that influence the numeric values chosen by people, see Section 2.7.1).

?

What is the distribution of numeric values in application domains? Figure 7.34 shows the yearly occurrence of number words (averaged over each year since 1960) in Google's book data. The English counts are larger because most of the books processed were written in English. Decade values (e.g., "ten", "twenty") follow their own trend, which is much higher than surrounding values.

Figure 13.3 shows that the distribution of numeric literals varies with the representation used to specify values.

Repeated occurrence of a small set of values during execution...

#### 7.3.6.2 Calls

Roughly 1-in-5 statements contains an explicit function/method call (compilers sometimes introduce additional calls to implement language constructs; see Figure 7.4).

How many distinct method calls occur within a project?

A study by Lämmel, Pek and Starek<sup>908</sup> investigated calls to methods from third-party APIs, and those defined within 1,435 projects. Figure 7.36 shows the number of distinct API methods called against project size (measured in method calls); the line is the fitted regression model:  $unique = e^{-1.5} calls^{0.78}$ .

The function/method called may be passed as an argument (i.e., a *callback*). A study by Gallaba, Mesbah and Beschastnikh<sup>538</sup> investigated the use of callbacks in 130 Javascript programs. Figure 7.37 shows the total number of calls in each program, against the number of calls containing callbacks and just anonymous callbacks; lines are the fitted regression

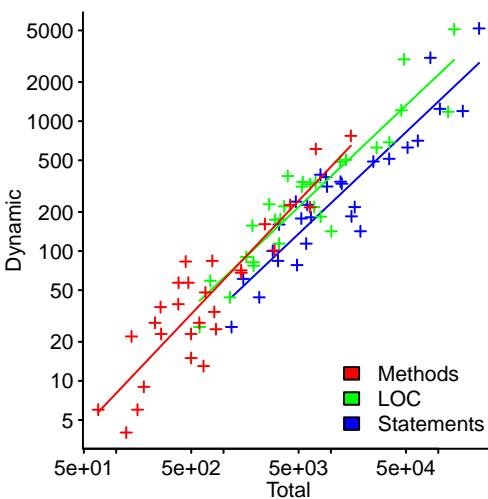


Figure 7.33: The number of dynamic statements, LOC and methods against total number of those constructs appearing in 28 Ruby programs; lines are power-law regression fits. Data from Rodrigues et al.<sup>1343</sup> [code](#)

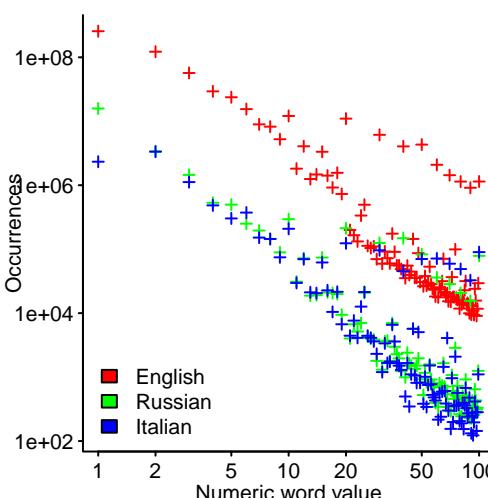


Figure 7.34: Yearly occurrence of numbers words (e.g., "one", "twenty-two"), averaged over each year since 1960, in Google's book data for three languages. Data kindly provided by Piantadosi.<sup>1246</sup> [code](#)

models:  $allCallbacks = e^{-0.8} allCalls^{0.86}$  and  $anonCallbacks = e^{-1.5} allCalls^{0.8}$ , respectively.

Some calls can be combined into a narrative sequence, e.g., open a file, write to it and then close it. Detecting some such narrative sequences is straightforward in object-oriented software; when a variable refers to an object, methods associated with the object can be called using the variable, e.g., `var.strLength()`.

A study by Mendez, Baudry and Monperrus<sup>1064</sup> investigated method sequences called on the same variable, based on an analysis of 4,888 classes in 3,418 Jar files (i.e., Java bytecode; some method calls may not appear explicitly in the original source, e.g., the compiler maps string concatenation using binary + to a call to the method `StringBuilder.append`). Figure 7.38 shows the 10 most frequent sequences of `java.lang.StringBuilder` methods called on the same variable (lines connect methods forming a sequence; method call argument types were ignored).

Figure 7.39 shows the number of sequences having a given length (i.e., measured in methods) and number of sequences that appear in the code (i.e., are used) a given number of times...; only classes whose method sequences are used at least 100 times are included.

### 7.3.6.3 Use of variables

Patterns in the use of variables is a rarely studied subject.

Many variables take one of approximately 10 roles.<sup>1371</sup> Roles include: *stepper* which systematically takes predictable successive values, *follower* which obtains its new value from the old value of another variable, and *temporary* which holds some value for a short time.

Some usage patterns are driven by the constructs available in the language used. For instance, in C loops often involve three accesses of the loop control variable, e.g., for (`i=0; i<10; i++`); while in languages that support constructs of the form for (`i` in `v_list`), only one access is required; in languages with vector operations loops may not explicitly appear in the source.

A study<sup>456</sup> of four large PHP applications found that less than 1% of variables were assigned values having different types (e.g., assigning an array and later assigning an integer).

An analysis<sup>312</sup> of integer usage in C found that around 20% of accesses to variables having an integer type were made in a context having a signedness that is different from the declared type; also the declaration of variables having an integer type are not usually modified.

### 7.3.7 Declarations

Declarations provide two basic services: a method of specifying the representation to be used for values, e.g., a variable's type, and a method of associating one or more names with stored values.

???

How do declarations change over time?

A study<sup>1140</sup> of the release history of a number of large C programs, over 3-4 years and a total of 43 releases, found that one or more fields were added to one or more existing structure or union types in 79% of releases, while structure or union types had one or more fields deleted in 51% of releases; another study<sup>7</sup> found one or more existing fields had their types changed in 35% of releases. [glob-var-usage.sxc](#)

A study by Robbes, Róthlisberger and Tanter<sup>1336</sup> investigated data extensions (i.e., the visitor pattern) and operation extensions to Smalltalk classes; the 2,505 projects analyzed contained 95,662 classes, forming 48,595 class hierarchies (with 41% containing more than one class). Figure 7.40 shows the number of data and operation extensions made to 1,560 class hierarchies containing both kinds of extension. The fitted regression model is:  $\log(data) = 0.91 + 0.14\log(operation)^2$ <sup>2</sup>.

In biological systems connections between components have a cost (e.g., they cause delays in a signalling pathway), and modularity is an organizational method that reduces the number of connections needed;<sup>305</sup> modularity as a characteristic that makes it easier to adapt more quickly when the environment changes may be an additional benefit, rather than the primary driver towards modularity. Simulations<sup>1065</sup> have found that, for non-trivial systems, a hierarchical organization reduces the number of connections needed (for a viable implementation).

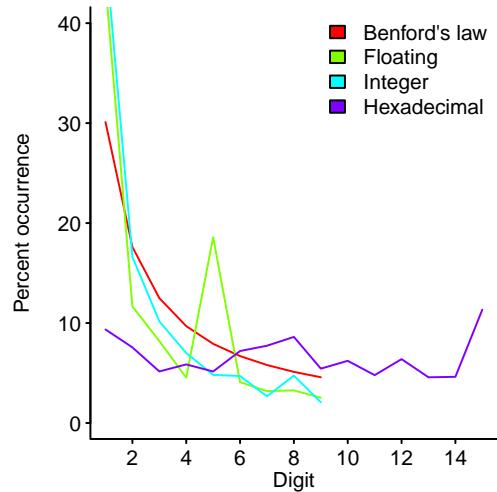


Figure 7.35: Percentage occurrence of the most significant digit of floating-point, integer and hexadecimal literals in C source code. Data from Jones.<sup>785</sup> [code](#)

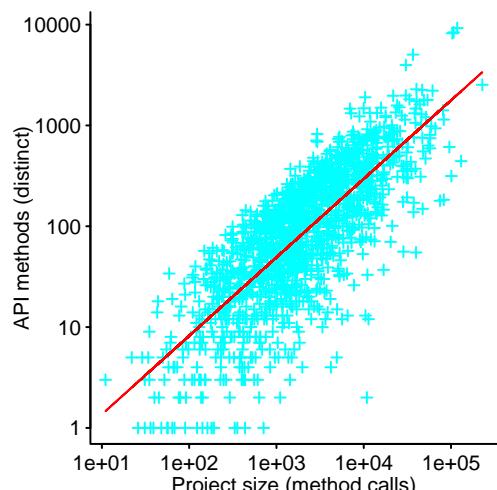


Figure 7.36: Number of distinct API methods called in 1,435 Java projects containing a given number of method calls, with regression fit. Data from Lämmel et al.<sup>908</sup> [code](#)

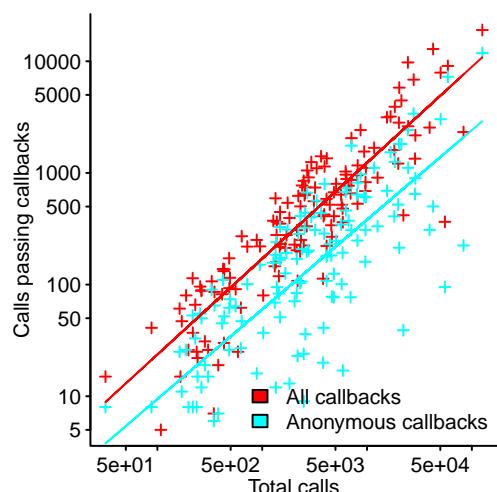


Figure 7.37: Number of function calls, against corresponding number of calls containing callbacks and anonymous callbacks, in 130 Javascript programs; lines are fitted regression models. Data from Gallaba et al.<sup>538</sup> [code](#)

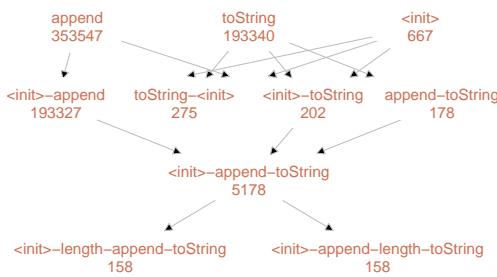


Figure 7.38: Sequences of methods, from `java.lang.StringBuilder`, called on the same object; based on 3,418 Jar files. Data from Mendez et al.<sup>1064</sup> [code](#)

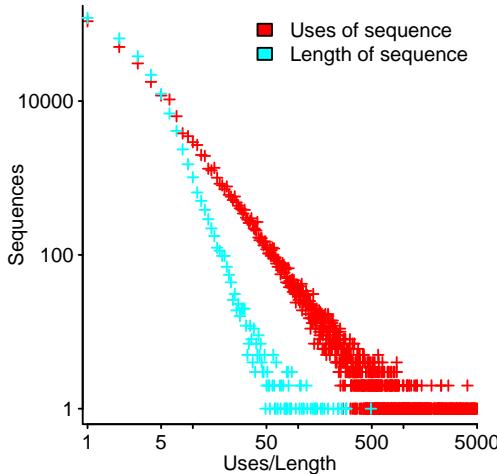


Figure 7.39: For each Java class, the percentage of method sequences containing a given number of calls, in 3,418 Jar files. Data from Mendez et al.<sup>1064</sup> [code](#)

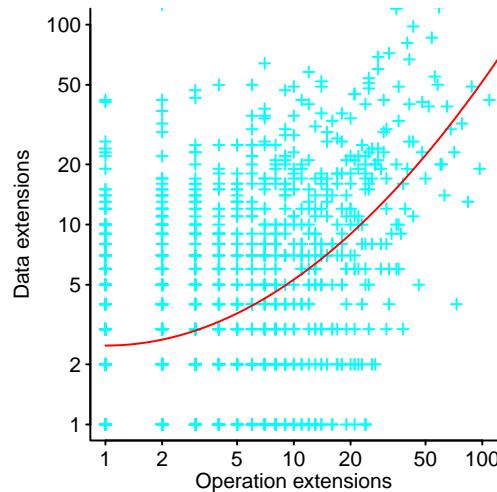


Figure 7.40: Number of data and operation extensions to 1,560 Smalltalk class hierarchies that contain both kinds of extension; line is a fitted regression model. Data from Robbes et al.<sup>1336</sup> [code](#)

### 7.3.7.1 Visibility

Large programs contain tens of thousands of identifier uses, and various language constructs have been created to allow developers to manage visibility and accessibility of declared identifiers, e.g., the private and public attributes in Java; R is an example of a language that provides no such functionality.

There are many reasons why identifiers may have greater visibility than necessary, e.g., classes and methods in Java<sup>1611</sup> (see [sourcecode/TR\\_DCC-overExposure/TR\\_DCC-overExposure.R](#)), including: at some point in the past greater visibility was required, developers are not aware of the situation, future plans...

### 7.3.7.2 Ordering of definitions in aggregate types

Patterns have been found in the ordering of declarations within Java class and C struct types; members sharing an attribute are often grouped together or have a preferred relative ordering.

These usage patterns may be the result of many developers making individual choices (either explicitly or implicitly), or similar ordering rules are specified (and followed), e.g., the Java coding conventions (JCC)<sup>1520</sup> specifies a recommended ordering for the declaration of: class variables (or fields), instance variables (or static initializers), constructors and methods (see Figure 12.7).

Section 12.4 discusses the analysis of items thought to have a preferred order.

Patterns in the ordering of fields in C struct types are discussed in Section 9.6.1. One interpretation of the pattern sound is reducing unused storage by placing together objects whose types have the same alignment requirements.

A study by Geffen and Maoz<sup>554</sup> investigated various patterns of method ordering within Java classes; for instance, the rules specified by the StyleCop tool (i.e., grouped by access modifiers), the commonly seen pattern of called methods appearing after the method that involved them, and the concept of connectivity (i.e., related methods appearing close to one another)...

A study by Biegel, Beck, Hornig and Diehl<sup>160</sup> investigated the impact of the kind of activity performed by a method on its relative ordering; the method activity attributes considered were: 1) all static methods (declared using the `static` keyword), 2) initializers (method name begins with `init`), 3) getters and setters (non-void return type and method name begins with `get`, `is` or `set` followed by a capital letter) and 4) all other non-static methods.

Figure 7.41 shows that methods performing two of the activities are very likely to be ordered before methods performing the remaining two other activities. Method declaration sequences containing more than two kinds of method activity occurred in 62% of contexts analysed, with 54% of these passing the threshold needed for analysis, leaving 33% of all declarations sequences.

The original author may not be responsible for maintenance, and it is possible that declarations added during maintenance were not inserted into an existing structure/class declaration according to the ordering pattern used during initial development; experience shows that some developers prefer to add new declarations at the end of an existing structure/class.

### 7.3.7.3 Unused variables

Source sometimes contains declarations of identifiers, which are not referenced outside of the declaration. Reasons for unused variables include: a mistake has been made (e.g., the variable should have been referenced), the variable was once referenced (i.e., the declaration is now redundant), and there is an expectation of future need for the variable.

Function parameters are an example where deleting the declaration of an unused variable may require a greater investment than the developer working on the function is willing to make, i.e., changing the number of function parameters requires corresponding changes to the arguments of all calls.

Figure 7.42 shows the total number of functions having a given number of parameters, a given number of unused parameters, and various fitted regression models. Unused function

parameters, which at around 11% of all parameters are slightly more common than unused local variables.

The fitted regression model, for the number of functions containing a given number of unused parameters is:  $functions = e^{4.6 - 0.5 \times unused}$ . In practice, a function is likely to acquire unused parameters one at a time, as the code evolves. An alternative formula for estimating the number of functions containing  $u$  unused parameters is:  $\sum_{p=u}^8 \frac{F_p}{7^p}$ , where:  $F_p$  is the total number of function definitions containing  $p$  parameters.

## 7.4 Building programs

Analysis of Makefile usage<sup>1022</sup>...?

size C/C++ systems, between 83% and 97% recompilations, specified in makefiles, were unnecessary<sup>1702</sup>...

The components of a system often need to be built in a specific way...? emailed...?

## 7.5 Evolution of source code

Software only changes when developers have an incentive to spend time making the changes. Incentives for developers to spend their time in this way include being paid, and a desire to change the code to satisfy a personal need (e.g., refactor code in the belief that other developers will have a higher opinion of the code...).

If payment is involved, there is a customer, and the changes are intended to address customer needs (it can be very difficult to work out what the customer needs actually are and there may be as many opinions about these needs as there are people working to keep the customer happy).

Software systems growth, in lines of code, over time is a commonly used metric; common reasons for growth include improvements to existing functionality and the addition of new functionality. Some systems grow at a consistent rate over many years, e.g., FreeBSD Figure 11.2, and the Linux kernel Figure 11.7, while others appear to have stopped adding lines, e.g., Figure 11.51 or grow sporadically, e.g., the Groovy compiler Figure 11.9.

Growth can also increase interdependencies between components; Figure 7.43 shows the relationship between the separate components of ANTLR over various releases.

The primary factors influencing the evolution of source code characteristics are:

- customer limited: insufficient customer demand (as measured by a willingness to pay) for it to be economically worthwhile updating existing functionality (to support changes in the world) or adding new functionality, e.g., new hardware requiring device drivers for a particular operating system,
- developer limited: bottlenecks in the development process that restrict the quantity of change per unit time. For instance, a limited number of people with the necessary skills, changes requiring sign-off by a handful of senior managers or increasing effort required to support a growing system leading to diminishing returns from adding more developers,
- some combination of customer and developer limits: variability of customer demand creates uncertainty about when and how much developer resource will be needed; resource scheduling... leading to a stop-go effect...

A consistent rate of growth suggests some degree of consistency in customer demand and the developer effort available to do the work...

Updating existing functionality can result in source code being deleted. The turnover of source code in Linux is evident in Figure 7.44, which shows the percentage of code in 130 releases that originated in earlier releases, and Figure 4.17 shows code shared between different releases of related BSD operating systems; Figure 11.67 shows the correlation between lines added/deleted for glibc, Figure 9.20 shows a Markov chain for file creation/modification/deletion in the Linux kernel...

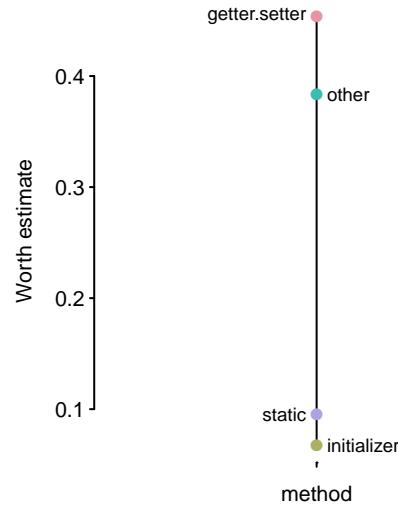


Figure 7.41: "Worth estimate" for the kind of method attribute (see Section 12.4). Data from Biegel et al.<sup>160</sup> code

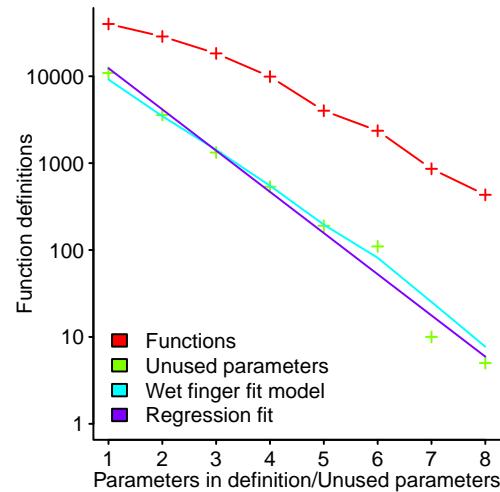
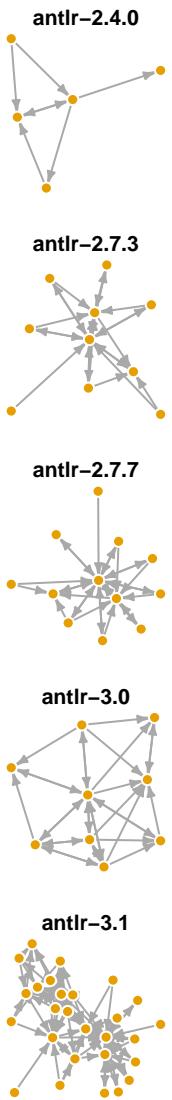


Figure 7.42: Number of C functions having a given number of unused parameters; straight lines are fitted regression models. Data from Jones.<sup>785</sup> code



[ecosystems/shi2013.R](#) or Volvo Masters... ?

emailed for more data... ?

kbuild and linux commits per day/week... Time series of commits to the Linux source and configurations files... ? Fig 3

Evolution of database frameworks... Table 11.7... ?

### 7.5.1 Database schema

Many applications depend on information contained in databases, with access requests often taking the form of SQL queries. The structure of a database, its schema, may change over time, e.g., columns are added/removed from tables and tables are added/removed; changes may be needed to support new functionality, or a reorganization (to save space or improve performance). Changes to a schema may entail changes to the SQL embedded in application code that accesses the database.

A database's contents may be used by many applications, and a change to the schema that impacts the behavior of existing applications may be costly. The cost may be high enough to outweigh the benefits of a schema change (assuming it is possible to know in advance, which applications are likely to be affected), making schema changes that could impact existing code unlikely.

A study by Skoulis<sup>1454</sup> investigated changes to the database schema of various projects over time; projects included Mediawiki (the software behind Wikipedia and other wikis), and Ensembl (a scientific project involving the European Bioinformatics Institute and the Wellcome Trust Sanger Institute). Figure 7.45 suggests that table growth is approximately linear in one case (just like source code growth currently encountered in many systems, e.g., Figure 11.2), and appears to have reached its maximum in another (e.g., Figure 11.51). Systems grow in response to customer requirements...

Figure 7.46 shows the table survival curve for the Mediawiki and Ensembl database schema. Why is the table survival rate for Wikimedia much higher than Ensembl? Perhaps there are more applications making use of the contents of the Wikimedia schema, and the maintainers of the schema don't want to generate discontent in their user-base, or the maintainers are just being overly conservative. Alternatively uncertainty over what data might be of interest in the Ensembl scientific project may result in the creation of tables that eventually turn out to be unnecessary and with only two institutions involved, table removal may be an easier decision. The only way of finding out what customer demands are driving the changes is to talk to those involved.

The presence of tables and columns in a schema does not mean they are used by applications; the database administrator may be unaware that they are unused, or they may have been inserted for use by yet to be written code.

A database may contain multiple tables, with information in different tables tied together using foreign keys. A study<sup>1598</sup> of the evolution of a six databases found wide variation in the handling of foreign keys.

Any changes in newly created tables over time? Number of tables of a given size created... changes in definition of existing columns...

??

In some applications the database schema are contained within string literals of languages such as PHP and Python,<sup>967</sup> which makes it more difficult to analyse them...

### 7.5.2 Source code lifetime

The survival rate of source code is a crucial input to any analysis of cost/benefit of investing to reduce future maintenance costs. Termination of source code usage occurs at various levels of granularity, from the top-level of the application or library containing it ceasing to be used, to the deletion of individual lines of code.

A function or method is a self-contained unit of code, and an obvious candidate for a cost/benefit analysis. Analyzing a sequence of lines, of code, is likely to be complicated because of

Figure 7.43: Dependencies between the Java packages in various versions of ANTLR. Data from Al-Mutawa.<sup>21</sup> code

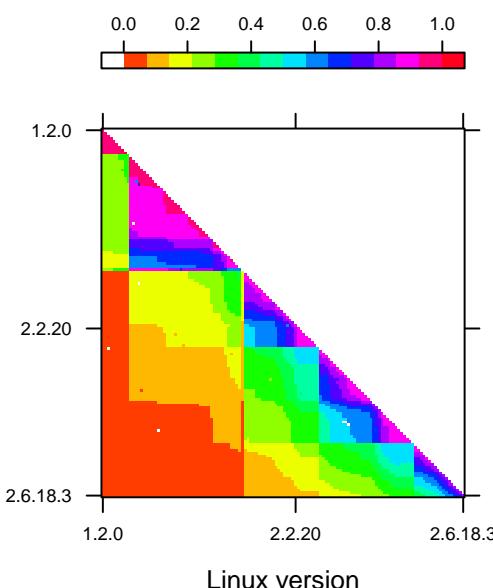


Figure 7.44: Fraction of source in 130 releases of Linux (x-axis) that originates in an earlier release (y-axis). Data extracted from png file kindly supplied by Matsushita.<sup>975</sup> code

the need to estimate the impact of the other locally associated lines (in many languages these would be contained within the same function).

A new function definition is about to be written, and it is thought that at some future time it may need to be modified. If an investment,  $I$ , in extra work is made to receive a benefit,  $B$ , during each future modification, what is the relationship between  $I$  and  $B$ ?

Like all investments, the expected benefit has to be greater than the investment:

$$I < M_t B$$

where  $M_t$  is the total number of modifications of the function.

Let  $0 < s < 1$  be the probability that a function will be modified in the future, and that once modified the likelihood of it being modified again remains unchanged; the expected number of modifications of a given function is then:

$$M_t = s + 2s^2 + 3s^3 + \cdots + ns^n$$

where  $n$  is the maximum number of modifications of a function; this series sums to:

$$M_t = \frac{s - (n+1)s^{n+1} + ns^{n+2}}{(1-s)^2}$$

substituting and rearranging the first equation, and assuming  $(n+1)s^{n+1}$  is very small, gives:

$$\frac{B}{I} > \frac{(1-s)^2}{s}$$

What range of values of  $s$  occur in practice? A study by Robles, Herraiz, German and Izquierdo-Cortázar<sup>1341</sup> analysed the change history of functions in Evolution (114,485 changes to functions over 10 years), and Apache (14,072 changes over 12 years).

Figure 7.47 shows the number of functions (in Evolution) that have been modified a given number of times (upper), and the number of functions modified by a given number of different authors (lower). A bi-exponential model provides a reasonable fit to both sets of data. One interpretation of a bi-exponential model in this context is that the processes driving the behavior have different characteristics for a few changes, or authors; perhaps a few changes/authors occur early in the history of a function, with subsequent changes happening much later...

The two exponentials, for number of changes gives  $s = 0.48$  ( $\frac{B}{I} > 0.58$  for Evolution, 0.47 for Apache), and  $s = 0.85$  ( $\frac{B}{I} > 0.03$ , 0.02 for Apache), the number of author fit gives  $s = 0.30$  ( $\frac{B}{I} > 1.6$  for Evolution, 3.1 for Apache), and  $s = 0.61$  ( $\frac{B}{I} > 0.24$ , 0.15 for Apache).

Is there less benefit in making an investment for the original author, do the benefits come from reducing the effort of subsequent authors in understanding the code?

The model does not include the benefit received when developers read the code without modifying it, assuming the investment was in ease of comprehension...

Figure 7.48 shows the number of modifications of a function, broken down by number of authors. this is multi-state censored data, need a fancier model... TODO

Functions are created, modified zero or more times and may be deleted before the complete package/module that contains them is deleted, or the application itself ceases to be used (??? in Evolution, ??? in Apache; many function definitions are never modified ??? in Evolution, ??? in Apache)...

A study by Wang<sup>1631</sup> looked at the survival of clones (a duplicate sequence of 50 or more tokens) in the Linux high/medium/low level SCSI subsystems (the architecture of this system happened to have three levels). Figure 5.42 shows the survival curves, which have an initial half-life of around 18 months, but the survival rate increases over time. A proxy for code lifetime...

?

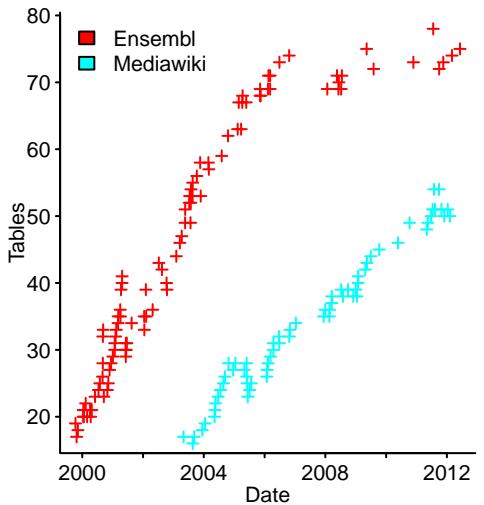


Figure 7.45: Changes in the number of tables in the Mediawiki and Ensembl project database schema over time. Data from Skoulis.<sup>1454</sup> [code](#)

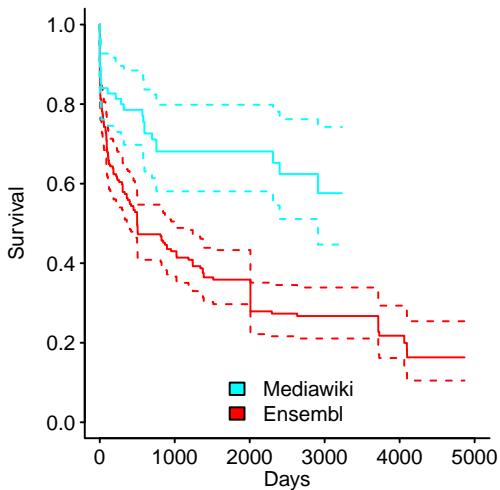


Figure 7.46: Survival curve for tables in Wikimedia and Ensembl database schema. Data from Skoulis.<sup>1454</sup> [code](#)

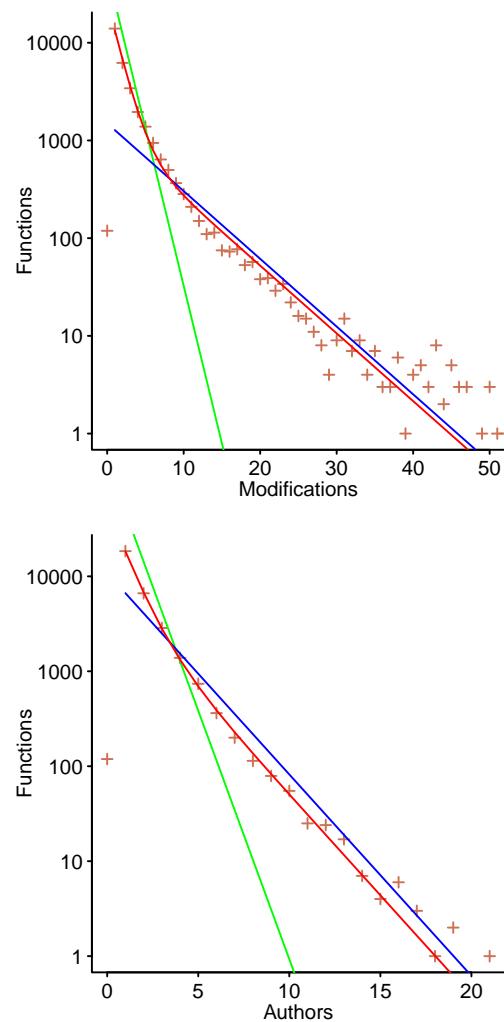


Figure 7.47: Number of functions (in Evolution; the point at zero are incorrect counts) modified a given number of times (upper) or modified by a given number of different people (lower); red line is a bi-exponential fit, green/blue lines are the individual exponentials. Data from Robles et al.<sup>1341</sup> [code](#)

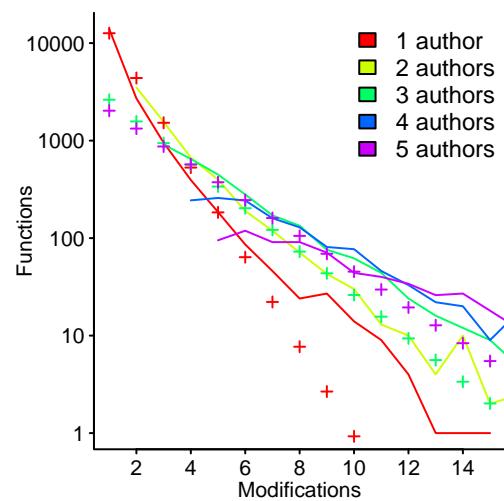


Figure 7.48: Number of functions (in Evolution) modified a given number of times broken down by number of authors. Data from Robles et al.<sup>1341</sup> [code](#)

# Chapter 8

## Stories told by data

### 8.1 Introduction

Data analysis is the process of finding patterns in data and weaving a story around these patterns.

Finding patterns in data is easy, weaving a believable narrative around them can be very difficult. Figure 8.1 may be interpreted as evidence for a causal connection between UFO activity and computer virus infections. Domain knowledge (e.g., personal experience of reporting problems and events) might lead us to believe that these reports were made by people, and an alternative interpretation is that counties with larger populations experienced and reported more virus infections and UFO sightings, compared to counties having smaller populations.

An understanding of the common patterns found in data is the starting point for an appreciation of the kinds of stories that these patterns might be used to substantiate. This chapter starts with an overview of techniques that may be used to uncover patterns in data, before moving on to discussing the communication of these patterns to others. Those performing the analysis are responsible for weaving a story around the patterns found; the figures, and numeric values provide props that may be used to conjure a convincing narrative.

The patterns sought have the form of a relationship between two or more of the measured quantities. Managers want to control software development and to do this they need understanding of the processes that are driving it. Regression modeling is this book's default technique for modeling the relationships between the quantities that have been measured; see Section 10.5.1.

Ideally you, the data analyst, have:

- enough domain knowledge to distinguish between spurious correlations that may be present in the data and correlations connected to the processes that generated the data,
- an idea of the questions for which answers are sought, in practice there may be a lot of uncertainty about what the questions are.

Questions have to have answers used can be used to make predicts about expected patterns of behavior in the data (which can be searched for).

If a question does not have an associated answer that has a predictable, detectable, pattern of behavior, then 42 is as good an answer as any other,

- the time and resources needed to obtain data likely to contain answers to the questions asked; obtaining data is often time-consuming and/or expensive and it is often necessary to make do with whatever data is cheaply and quickly available (even if it only indirectly relate to the questions being asked). This book generally assumes that a dataset has been obtained, some of the issues around obtaining data are discussed in Chapter 13.

The data should contain as little noise, in practice the available data may be very noisy and cleaning may be very time-consuming,

- statistical analysis techniques capable of providing answers to the desired level of certainty; in practice it may not be possible to draw any meaningful conclusions from the data or more questions will be uncovered.

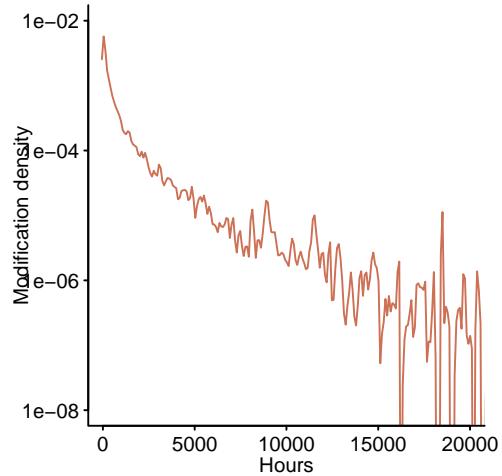


Figure 7.49: Density plot of time interval, in hours, between each modification of a function in Evolution. Data from Robles et al.<sup>1341</sup> [code](#)

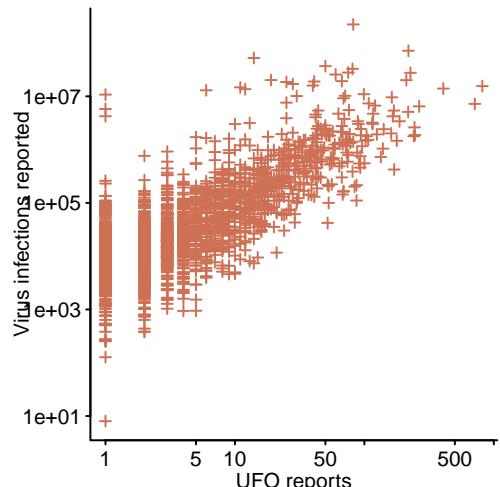


Figure 8.1: Number of virus infections and UFO sighting, reported in 3,072 U.S. counties during 2010. Data from Jacobs et al.<sup>766</sup> [code](#)

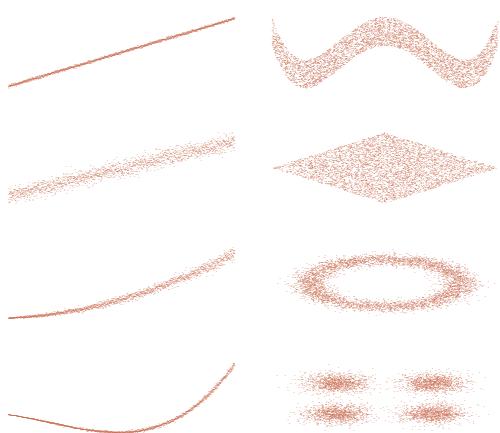


Figure 8.2: Data having values following various visual patterns when plotted. [code](#)

Data analysis is like programming, in that people get better with practice; there are a few basic techniques that can be used to solve many problems and doing what you did on a previous successful project can save lots of time.

This, and subsequent chapters explicitly discuss the R code that was used (previous chapters discuss analysis results, not analysis implementation).

There is no guarantee that the available data contains any information that might be used to answer any of the questions being asked of it.

Techniques for evaluating possible interpretations of any patterns found in data include:

- model simplicity...
- consistency with existing models of how things are believed to work...
- how well a model fits the available data...

If the data does not contain population information (and it cannot be easily obtained), the extent to which this alternative interpretation is consistent with the report data can be checked. Without appropriate data, alternative interpretation is based on the analysts model of the world from which the data was obtained.

At a bare minimum, the story told by an analysis of data needs to meet the guidelines for truthfulness in advertising that is specified by the national advertising standards' authority. If manufacturers of soap powder have to meet these requirements, when communicating with the public, then so should you...

**Check assumptions derived from visualizations** Assumptions suggested by a visualization of data need to be checked statistically. For instance, Figure 8.3 shows professional software development experience, in years, of subjects taking part in an experiment using a particular language. The visual appearance suggests that as a group, the PHP subjects are more experienced than the Java subjects. However, a permutation test, comparing years of experience for the PHP and Java developers, shows that the difference in mean values is not significant (there are only nine subjects in each group and the variation in experience is within the bounds of chance; see [communicating/postmortem-answers.R](#)).

## 8.2 Finding patterns in data

When a specific pattern is expected, the data can be checked to see whether it contains this pattern. Otherwise, the search for patterns is essentially a fishing expedition.

Figure 8.2 shows some common and less common patterns seen in data. The left column shows data forming lines of various shapes; a straight line is perhaps the most commonly encountered pattern in data and points may all be close to the line or form a band of varying width. The right column shows data clustering together in various regular shapes. Uncovering a pattern is the next step along the path to understanding the processes that generated the sample measurements.

Vision is the primary pattern detection pathway used in this book. Animals have developed sophisticated visual pattern detection and recognition systems (see Section 2.3); number processing is a very new ability, and as such is relatively slow and unsophisticated.

**Compelling numbers.** For small quantities of numeric data, the pattern present in the printed form of the values may be the most compelling visual representation. For instance, relative spacing is sometimes used within the visible form of expressions to highlight the relative precedence of binary operators (e.g., more whitespace around the addition operator when it appears adjacent to a multiplication, e.g.,  $5 + 2*3$ ). Table 8.1 shows that when relative spacing is used, it nearly always occurs in a form that where the operator with higher precedence has closer proximity to its operands (relative to the operator having a lower precedence). The number of cases where the reverse occurs is small, suggesting that either the developer who wrote the code did not know the correct relative precedence or there is a fault in the code.

A study by Landy and Goldstone<sup>914</sup> found that subjects were more likely to give the correct answer (and answer more quickly) to simple arithmetic expressions, containing two binary operators, when there was greater visual proximity between the operands that were separated by the binary operator having the higher precedence.

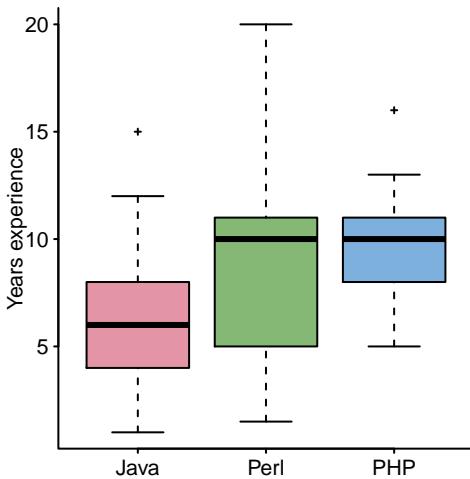


Figure 8.3: Years of professional experience in a given language for experimental subjects. Data from Prechelt.<sup>1278</sup> [code](#)

	Total	High-Low	Same	Low-High
<b>no-space</b>	34,866	2,923	29,579	2,364
<b>space no-space</b>	4,132	90	393	3,649
<b>space space</b>	31,375	11,480	11,162	8,733
<b>no-space space</b>	2,659	2,136	405	118
<b>total</b>	73,032	16,629	41,539	14,864

Table 8.1: Number of expressions containing two binary operators having the specified spacing in the visible source (i.e., no spacing, no-space, or one or more whitespace characters (excluding newline), space) between a binary operator and both of its operands. The High-Low column lists counts for expressions where the first operator of the pair has the higher precedence (some are expressions where the both operators of the pair have the same precedence), the Low-High column lists counts for expressions where the first operator of the pair has the lower precedence. For instance,  $x + y^*z$  is space no-space because there are one or more space characters either side of the addition operator and no-space either side of the multiplication operator, the precedence order is Low-High. Data from Jones.<sup>785</sup>

## 8.2.1 Initial data exploration

Initial data exploration starts with the messy issue of how the data is formatted (lines containing a fixed number of delimited values is the ideal form, because many tools accept this as input; if a database is provided it may be worth extracting the required data into this form).

A programmer's text editor is as good a tool as any for an initial look at data, unless the filename suggests it is a known binary format (e.g., spreadsheet or database). For data held in spreadsheets exporting the required values to a csv file is often the simplest solution.

This initial look at the data will reveal some basic characteristics, such as: number of measurement points (often the number of lines) and number of attributes measured (often the number of columns), along with the kind of attributes recorded (e.g., date, time, lines of code, language, cost estimated, email addresses, etc).

The most important reason for viewing the file with an editor, first, is to identify the character used to delimit columns.

A call to `read.csv` reads the entire contents of a text file into a data frame (what R calls a structure or record type). The file is assumed to contain rows of delimited values (there is an option to change the default delimiter); spurious characters or missing column entries can cause subsequent values to appear in the incorrect column (Chapter 14 provides some suggestions for finding and correcting problems such as this). The `foreign` package contains functions for reading data stored in a variety of proprietary binary forms.

Having read the file into a variable, the following functions are useful for forming an initial opinion of the characteristics of the data that has been read (unless the dataset is small enough to be displayed on a screen in its entirety):

- `str` returns information about its argument, e.g., the number of rows and columns, along with the names, types and first few values of each column in a data frame,
- `head` and `tail` print six rows from the start/end of their argument respectively,
- `table` prints a count of the number of occurrences of each value in its argument, e.g., a particular column of a `data.frame` (by default NAs are not included). The `cut` function can be used to divide the range of its argument into intervals and return the bounds of the intervals and the corresponding counts in each interval.

If `str` reports a column having an unexpected type (e.g., `chr` rather than `int`), the likely causes are missing values and spurious characters in the data.

When one or two columns are of specific interest, `plot` can be used to quickly visualize the specific values of interest. Chapter 14 discusses techniques for cleaning data.

The upper plot in Figure 8.4 shows a very noticeable change in the number of occurrences, in C source files, of lines having a given length (i.e., number of characters on a line). What might cause this pattern to occur?

The change occurs at around the maximum line length commonly supported by non-GUI, non-flat screen, terminals (these measurements are of C source that is over 10 years old, i.e., before flat screen monitors became available). One hypothesis is that a system limit has a significant impact on the usage characteristics. A prediction derived from this hypothesis is that code written by developers using terminals that supported more characters per line

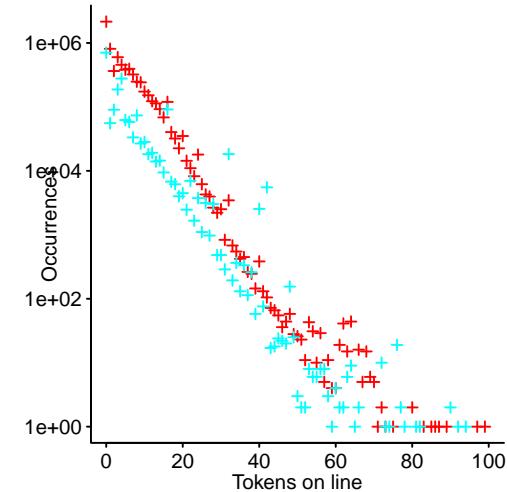
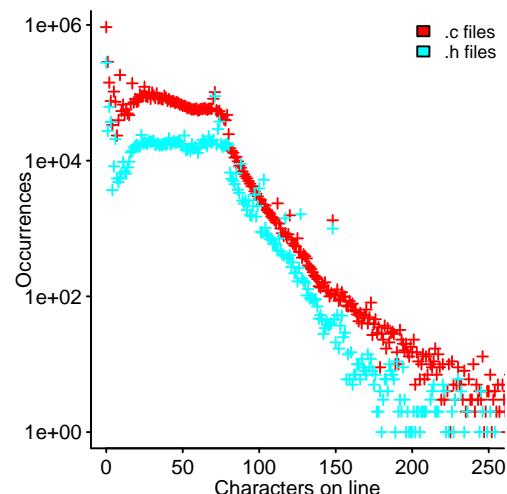


Figure 8.4: Total number of lines of C source, in .c and .h files, having a given length, i.e., containing a given number of characters (upper) and tokens (lower). Data from Jones.<sup>785</sup> code

would contain a greater number of longer lines, i.e., the downturn in the plot would move to the right.

The lower plot in Figure 8.4 illustrates that a different representation of the same information may not have any immediately obvious visual pattern. This plot is a count of the number of tokens per line. Knowing that average token length is around 3-4 characters, suggests that the slight change in the downward slope of the data points just visible at around 25 tokens corresponds to the more dramatic dip seen in the characters-per-line plot.

When a data set contains many variables, plotting one pair of variables at a time is an inefficient use of time. The plot, when given a data frame containing three or more columns, creates nested plots of every pair of columns. Figure 8.5 shows four sets of measurements relating to the same task; some measurement pairs are in a roughly linear relationship, while no obvious visual pattern is apparent for other pairs.

```
work=read.csv(paste0(ESUR_dir, "communicating/pub-fs-fp.csv.xz"), as.is=TRUE)
# -1 removes the first column
plot(work[, -1], col=point_col, cex.labels=1.2)
```

A list of columns can be specified using the formula notation; the following code has the same effect as the previous example:

```
plot(~ CFP+Haskell+Abstract+C, data=work[, -1], col=point_col, cex.labels=1.3)
```

If a more tailored visualization of pairs of columns is required, the `pairs` function supports a variety of options. For instance, separating out and highlighting subsets of a sample (known as *stratifying*) can be used to highlight differences and similarities. Figure 8.6 separates out measurements of Ada and Fortran projects. The lines are from fitting the points using loess, a regression modeling technique (see below and Section 11.2.5).

```
panel.language=function(x, y, language)
{
  fit_language=function(lang_index, col_str)
  {
    points(x[lang_index], y[lang_index], col=pal_col[col_str])
    lines(loess.smooth(x[lang_index], y[lang_index], span=0.7), col=pal_col[col_str])
  }

  fit_language(language == "Ada", 2)
  fit_language(language != "Ada", 1)
}

# rows 28 and 30 are zero, and we only want columns 16:19
pairs(log(nasa[-c(28, 30), 16:19]), cex.labels=1.3,
      panel=panel.language, language=nasa$language)
```

The default behavior of `pairs` produces a plot containing redundant information; it is possible to display different information in the upper and lower halves of the plot, and along the diagonal. Figure 8.7 shows expert and novice performance (time taken to complete various tasks and final test coverage) in a test driven development task, with a boxplot along the diagonal and correlation between each pair of attributes, for the two kinds of subjects, in the lower half of the plot. This plot, which primarily uses the default values for its visual appearance, needs more work before being presented to customers.

```
panel_user=function(x, y, user)
{
  expert=(user == "e")
  points(x[expert], y[expert], col=pal_col[1])
  points(x[!expert], y[!expert], col=pal_col[2])
}

panel_correlation=function(x, y, user)
{
  expert=(user == "e")
  r_ex=cor(x[expert], y[expert])
  r_nov=cor(x[!expert], y[!expert])
  txt = paste0("e= ", round(r_ex, 2), "\n", "n= ", round(r_nov, 2))
  text(0.1, 0.5, txt, pos=4, cex=1.6)
```

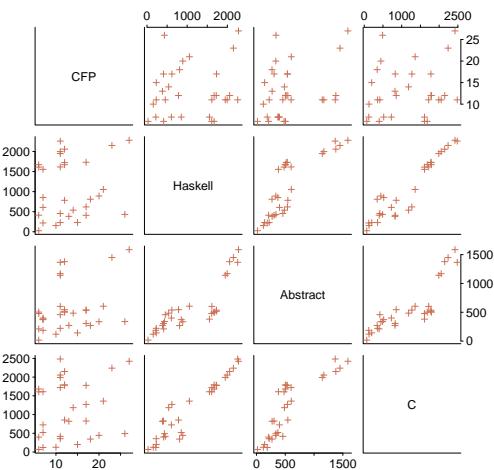


Figure 8.5: Various measurements of work performed implementing the same functionality, number of lines of Haskell and C implementing functionality, CFP (COSMIC function points; based on user manual) and length of formal specification. Data kindly provided by Staples.<sup>1490</sup> code

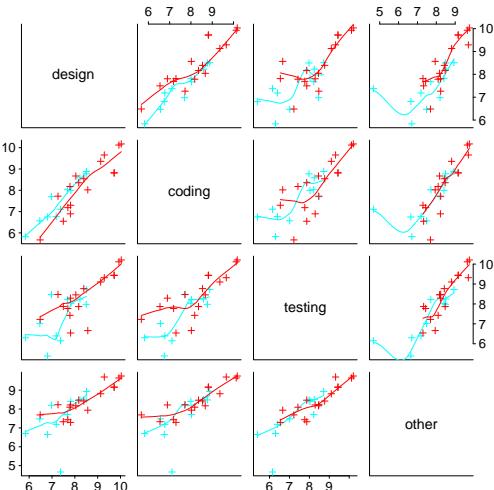


Figure 8.6: Effort, in hours (log scale), spent in various development phases of projects written in Ada (blue) and Fortran (red). Data from Waligora et al.<sup>1627</sup> code

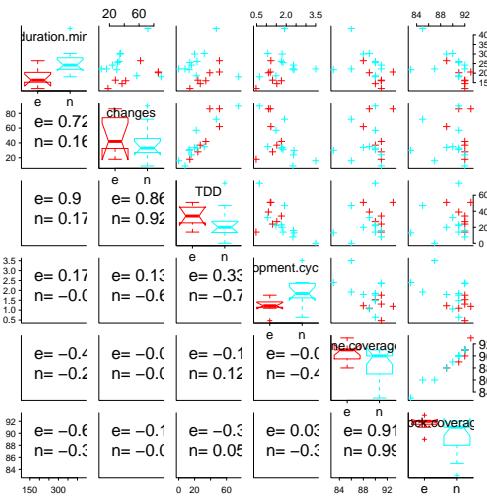


Figure 8.7: Performance of experts (e) and novices (n) in a test driven development experiment. Data from Muller et al.<sup>1116</sup> code

```

}

panel_boxplot=function(x, user)
{
t=data.frame(x, user)
boxplot(x ~ user, data=t, notch=TRUE, border=pal_col, add=TRUE)

}

pairs(~ duration.min+changes+TDD+
      log(development.cycle.length)+line.coverage+block.coverage,
      data=tdd, cex.labels=1.3,
      upper.panel=panel_user, lower.panel=panel_correlation,
      diag.panel=panel_boxplot, user=tdd$user)

```

More complex pair-wise plots can be created by the `splom` function in the `lattice` package.

As the number of columns increases, the amount of detail visible in a `pairs` plot decreases. The correlation between pairs of columns can be compactly displayed and provides the minimally useful information (i.e., a linear relationship exists).

The `corrgram` package supports a variety of functions for displaying correlation information. Figure 8.8 shows the correlation between every pair of 27 columns, with correlation used to control the color and eccentricity of each entry; with blue/clockwise denoting a positive correlation and red/anti-clockwise a negative one.

Having column names appear along the diagonal creates a compact plot; when many columns are involved this form of display is better suited to situations where the names follow a regular pattern. The `plotcorr` function in the `ellipse` package places column names around the outside (see [communicating/pull-req-cor.R](#)).

```

library("corrgram")

corrgram(ctab, upper.panel=panel.pie, lower.panel=panel.shade)

```

Hierarchical clustering is another technique for finding columns that share some degree of similarity, based on a user supplied distance metric. The `hclust` function requires the user to handle the details; the `varclus` function in the `Hmisc` package provides a higher level interface. The following code uses `as.dist` to map the cross-correlation matrix returned by `cor` to a distance; see Figure 8.9:

```

ctab = cor(used, method = "spearman", use="complete.obs")
pull_dist=as.dist((1-ctab)^2)

t=hclust(pull_dist)
plot(t, main="", sub="", xlab="Pull related variables", ylab="Height\n")

```

## 8.2.2 Guiding the eye through data

It may be difficult to reliably estimate the path of a central line through a collection of points, in a plot (according to some other goodness of fit criteria; Section 11.2.5 contains a more detailed discussion of fitting a trend line to data). A way to quickly add such a line to an existing plot is to use the `loess.smooth` function.

```

plot(res_tab$Var1, res_tab$Freq, col=pal_col[2],
     xlab="SPECint result", ylab="Number of computers\n")

lines(loess.smooth(res_tab$Var1, res_tab$Freq, span=0.3), col=pal_col[1])

scatter.smooth(res_tab$Var1, res_tab$Freq, span=0.3, col=point_col,
              xlab="SPECint Result", ylab="Number of computers\n")

```

The `scatter.smooth` function both plots and draws the loess line (no options are available to control the color of the line).

Plotting a fitted line is a way of visually showing that expectations of a pattern of behavior is being followed (or not). Figure 8.11 shows a loess fit (green) to NASA data<sup>635</sup> on cost

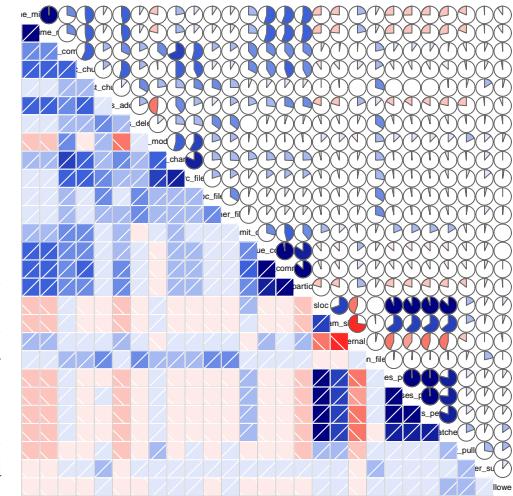


Figure 8.8: Correlations between pairs of attributes of 12,799 Github pull requests to the Homebrew repo, represented using pie charts and shaded boxes. Data from Gousios et al.<sup>605</sup> [code](#)

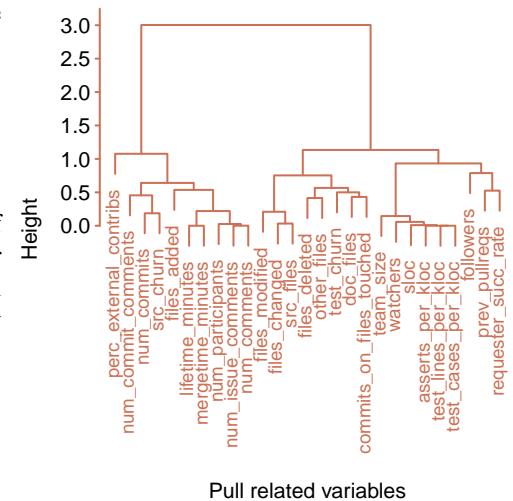


Figure 8.9: Hierarchical cluster of correlation between pairs of attributes of 12,799 Github pull requests to the Homebrew repo. Data from Gousios et al.<sup>605</sup> [code](#)

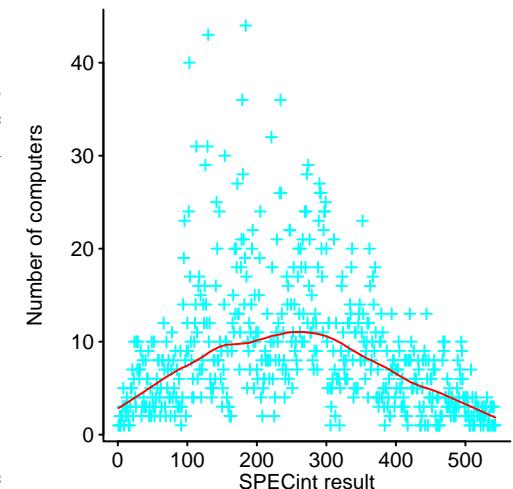


Figure 8.10: Number of computers having a given SPECint result; line is a loess fit. Data from SPEC.<sup>1471</sup> [code](#)

overruns for various space probes, against effort invested in upfront project definition; the upward arrow shows the continuing direction of the line seen in the original plot created by one user of this data (who was promoting a message that less investment is always bad).

There are a variety of techniques for calculating a smooth line that is visually less noisy than drawing a line through all the points. Splines are invariably suggested in any discussion of fitting a smooth curve to an arbitrary set of points; the `smooth.spline` function will fit splines to a series of points and return the x/y coordinates of the fitted curve.

Splines originated as a method for connecting a sequence of points by a visually attractive smooth curve, not as a method of fitting a curve that minimises the error in some measurement. LOESS is a regression modeling technique for fitting a smooth curve that minimises the error between the points and the fitted curve; the `loess.smooth` function fits a loess model to the points and return the x/y coordinates of the fitted curve.

Calculating missing values by extrapolation... Buettner staffing over time...

Both splines and loess can be badly behaved when asked to fit points that include extreme outliers or have regions that are sparsely populated with data. The running median (e.g., `median(x[1:k])`, `median(x[(1+1):(k+1)])`, `median(x[(1+2):(k+2)])`) and so on for some k) is a smoothing function that is robust to outliers; the `runmed` function calculates the running median of the points and return these values (the points need to be in increasing, or decreasing, order).

Figure 8.12 shows the maximum clock frequency of cpus introduced between 1971 and 2010; the various lines were produced using the values returned by the `smooth.spline`, `loess.smooth` and `runmed` functions. Don't be lulled into a false sense of security by the lines looking very similar, the *smoothing parameter* provided by each function was manually selected to produce a pleasing looking fit in each case; the mathematics behind the functions can produce curves that look very different and the choice of function will depend on the kind of curve required and perhaps be driven by the characteristics of the data.

```
plot(x_vals, y_vals, log="y", col=point_col,
      xlab="Date of cpu introduction", ylab="Relative frequency increase\n")
lines(loess.smooth(x_vals, y_vals, span=0.05), col=pal_col[1])

# smooth.spline and runmed don't handle NAs
t=!is.na(x_vals) ; x_vals=x_vals[t] ; y_vals=y_vals[t]
t=!is.na(y_vals) ; x_vals=x_vals[t] ; y_vals=y_vals[t]

lines(smooth.spline(x_vals, y_vals, spar=0.7), col=pal_col[2])

t=order(x_vals)
lines(x_vals[t], runmed(y_vals[t], k=9), col=pal_col[3])
```

Lines drawn through a sample of measurements values often follow the path specified by a central location metric, e.g., the mean value. In more cases it may be more informative to fit a line such that 25% of the measurements are below/above it, or some other percentage; *quantile regression* is a popular technique used for fitting such lines. Figure 8.13 is based on a study<sup>1342</sup> of 2,183 replies from a survey of FLOSS developers; two of the questions being the year and age at which those responding first contributed to FLOSS.

If you find yourself writing lots of algorithmic R code during initial data exploration, you are either investing too much effort in one area or you have found what you are looking for and have moved past initial exploration. Why are you writing lots of R, there is probably a package that does most of what you want to do and perhaps even more.

### 8.2.3 Smoothing data

Measured values sometimes fluctuate widely around a general trend (the data is said to be *noisy*). Smoothing the data can make it easier to see any pattern that might be present in the clutter of measured values. The traditional approach is to divide the range of measurement values into a sequence of fixed width bins and count the number of data points in each bin; the plotted form of this binning process is known as a *histogram*.

Histograms have the advantage of being easy to explain to people who do not have a mathematical background and existing widespread usage means that readers are likely to have

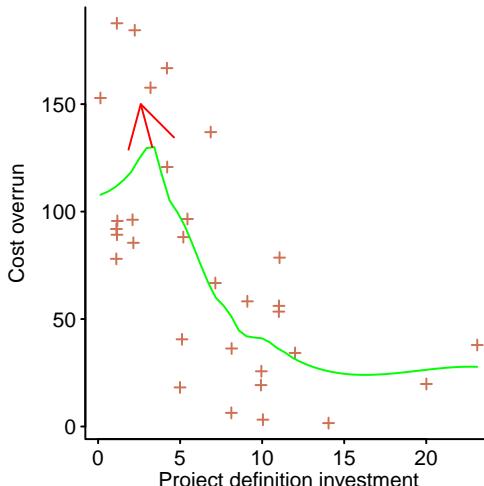


Figure 8.11: Effort invested in project definition (as percentage of original estimate) against cost overrun (as percentage of original estimate). Data extracted from Gruhl.<sup>635</sup> [code](#)

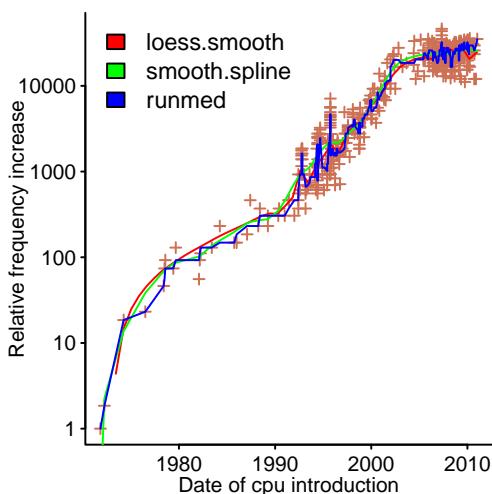


Figure 8.12: Relative clock frequency of cpus when first launched (1970 == 1). Data from Danowitz et al.<sup>356</sup> [code](#)

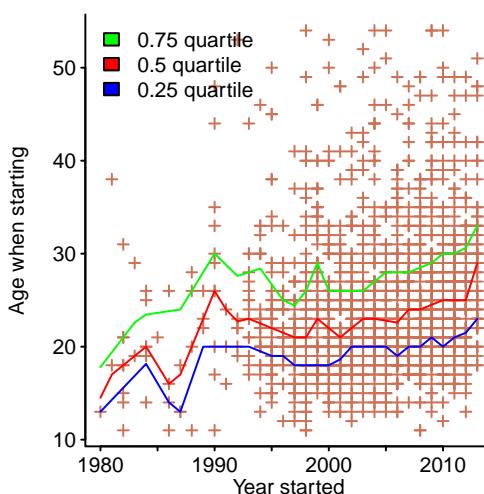


Figure 8.13: Year and age at which survey respondents started contributing to FLOSS, i.e., made their first FLOSS contribution. Data from Robles et al.<sup>1342</sup> [code](#)

encountered them before. Until the general availability of computers, histograms also had the advantage of keeping the human effort needed to smooth data within reasonable limits.

The upper plot of Figure 8.14 shows a count of computers having the same SPECint result, aggregated into 13 fixed width bins (the number of bins selected by the `hist` function for this data).

```
hist(cint$Result, main="", col=point_col,
     xlab="SPECint result", ylab="Number of computers\n")
```

The histogram package supports a wider range of functionality and more options than is available in the base system functions.

The advantage of the binning approach to smoothing and aggregating data is that it is an easy to perform manual task and for this reason it has a long history. The disadvantages of histograms are: 1) changing the starting value of the first bin can dramatically alter the visual outline of the created histogram, and 2) they do not have helpful mathematical properties.

A technique that removes the arbitrariness of histogram bins' starting position is averaging over all starting positions, for a given bin width (known as a *average shifted histogram*); this is exactly the effect achieved using kernel density with a rectangular kernel function.

It often makes sense for the contribution made by each value to be distributed across adjacent measurement points, with closer points getting a larger contribution than those further away. This kind of smoothing calculation is too compute-intensive to be suited to manual implementation, but are easily calculated when a computer is available.

The distribution of values across close measurement points is known as *kernel density*; histograms are the manual labourer's poor approximation to Kernel density, if a computer is available use the better technique.

The `density` function returns a kernel density estimate (which can be passed to `plot` or `lines`); the following code produced the lower plot in Figure 8.14:

```
plot(density(cint$Result))
```

Density plots also perform well when comparing rapidly fluctuating measurements of related items. The upper plot in Figure 8.15 shows the number of commits of different lengths (in lines of code) to the Linux filesystem code, for various categories of changes; the lower plot is a density plot of the same data.

The kernel density approach generalizes to more than one dimension; see the `KernSmooth` and `ks` packages.

When dealing with measurements that span several orders of magnitude, a log scale is often used. Creating a histogram using a log scale requires the use of bin widths that grow geometrically (coding is needed to get the `hist` function to use variable width bins; the `histogram` package contains built-in support for this functionality) and bin contents has to be expressed as a density (rather than a count). A histogram based on counts, rather than density, can produce misleading results; Figure 8.16 was produced by the following, where `y` is assigned decreasing, power law, values (the histogram should be continuously decreasing and not show a second peak):

```
x=1:1e6
y=trunc(1e6/x^1.5)
log_y=log10(y)

hist(log_y, n=40, xlim=c(0, 3),
     main="", xlab="log(quantity)", ylab="Count\n")
```

## 8.2.4 Densely populated measurement points

Some samples contain data having characteristics that result in plots containing lots of ink and little visual information. Common characteristics of the density of values, such as the following:

- adjacent values on the x-axis having widely different values on the y-values, e.g., Figure 8.10,

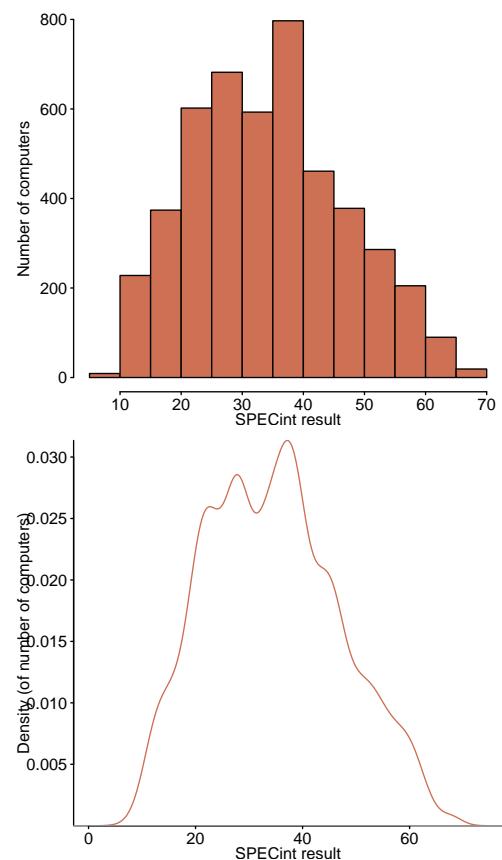


Figure 8.14: Number of computers with a given SPECint result, summed within 13 equal width bins (upper) and kernel density plot (lower). Data from SPEC. <sup>1471</sup> code

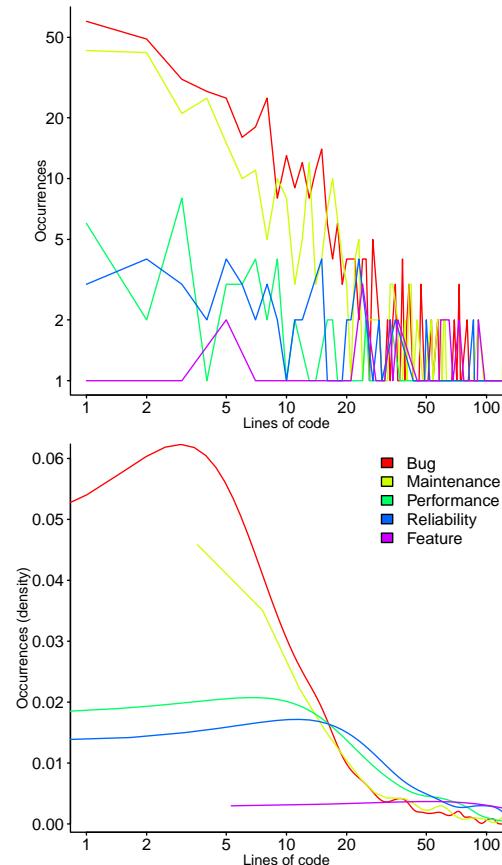


Figure 8.15: Number of commits containing a given number of lines of code made when making various categories of changes to the Linux filesystem code (upper) and a density plot of the same data (lower). Data from Lu et al. <sup>981</sup> code

- multiple points having the same x/y value, all combined visually as a single point in a plot, e.g., Figure 8.17,
- many very similar values that merge into a formless mass, when plotted, e.g., Figure 8.18.

A plot of values gives a misleading impression when multiple measurements have the same value, i.e., a single point represents many measurements (the problem is more likely to occur for measurements that can only take a small set of values, e.g., discrete values); see top plot in Figure 8.17. The jitter function returns its argument with a small amount of added random noise; the middle plot of Figure 8.17 shows the effect of jittering the values used in the upper plot. Another possibility is for the size of the plotted symbol to vary with the number of measurements at a given point (lower plot of Figure 8.17); as discussed elsewhere, people are poor at estimating the relative area and so size should not be treated as anything more than a rough indicator.

```
plot(maint$est_time, maint$act_time, col=point_col, xlab="",  
      ylab="Actual hours\n")  
  
plot(jitter(maint$est_time), jitter(maint$act_time), col=point_col,  
      xlab="Estimated hours", ylab="Actual hours\n")  
  
library("plyr")  
t=ddply(maint, .(est_time, act_time), nrow)  
plot(t$est_time, t$act_time, cex=log(1+t$V1), pch=1, col=point_col,  
      xlab="", ylab="Actual hours\n")
```

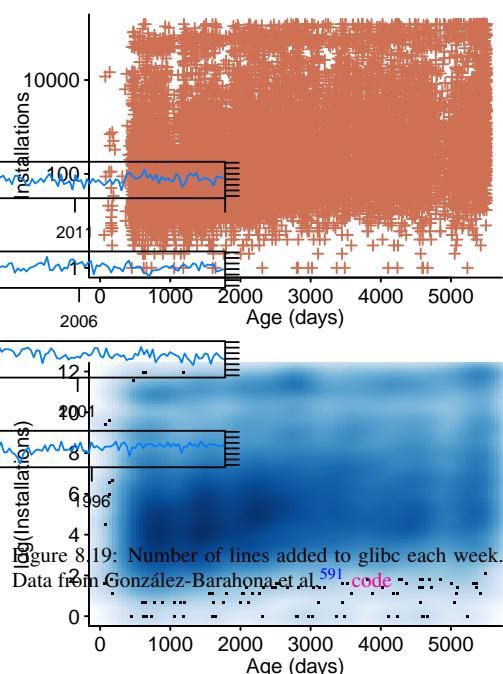
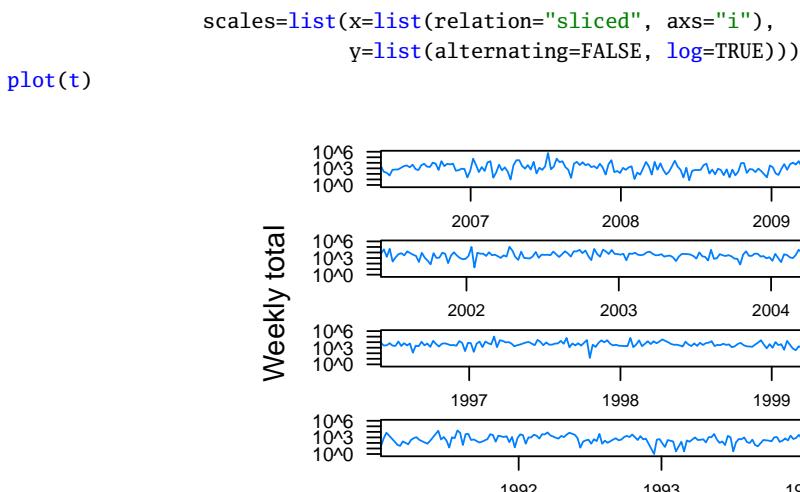
A different kind of communications problem occurs when data points are so densely packed together, that any patterns which might be present are hidden by the visual uniformity (see upper plot in Figure 8.18). One technique for uncovering patterns in what appears to be a uniform surface is to display the density of points. The smoothScatter function calculates a kernel density over the points to produce a color representation (middle plot); contour lines can be drawn with contour using the 2-D kernel density returned by kde2d (lower plot).

```
plot(udd$age, udd$insts, log="y", col=point_col,  
      xlab="Age (days)", ylab="Installations\n")  
# Bug in support for log argument :-(  
smoothScatter(udd$age, log(udd$insts),  
      xlab="Age (days)", ylab="log(Installations)\n")  
  
library("MASS")  
  
plot(udd$age, udd$insts, log="y", col=point_col,  
      xlab="Age (days)", ylab="Installations\n")  
  
# There is no log option, so we have to compress/expand ourselves.  
d2_den=kde2d(udd$age, log(udd$insts+1e-5), n=50)  
contour(d2_den$x, exp(d2_den$y), d2_den$z, nlevels=5, add=TRUE)
```

The hexbin package is available for those who insist on putting values into bins, in this case using hexagonal binning to support two dimensions.

One solution to a high density of points in a plot, is to stretch the plot over multiple lines; the xyplot function, in the lattice package, can produce a strip-plot such as the one in Figure 8.19, produced by the following code:

```
library("lattice")  
library("plyr")  
  
cfl_week=ddply(cfl, .(week),  
      function(df) data.frame(num_commits=length(unique(df$commit)),  
                            lines_added=sum(df$added),  
                            lines_deleted=sum(df$removed)))  
  
# Placement of vertical strips is sensitive to the range of values  
# on the y-axis, which may have to be compressed, e.g., sqrt(...).  
t=xyplot(lines_added ~ week | equal.count(week, 4, overlap=0.1),  
      cfl_week, type="l", aspect="xy", strip=FALSE,  
      xlab="", ylab="Weekly total",
```



## 8.2.5 Visualizing a single column of values

Sometimes the data contains a single column of values, or contains a column of interest that is not related to any of the other columns (i.e., there is no related column that can be used to create a 2-D plot).

A *box-and-whiskers* plot (or *boxplot* as it is more generally known) is a commonly used visualization for the distribution of a list of values. In Figure 8.20 the following characteristics are highlighted:

- median, i.e., the point that divides the number of values in half,
- first/third or lower/upper quartile, the 25th/75th percentiles respectively,
- lower/upper hinges, the points at a distance  $\pm 1.5 \cdot IQR$  where  $IQR$  is the interquartile range (the difference between the lower quartile and the upper quartile). The dotted line joining the hinges to the quartile box are the whiskers,
- outliers, all points outside the range of the lower/upper hinge.

The boxplot function produces a boxplot, and the argument notch=TRUE creates a plot that includes a *notch* indicating the 95% confidence interval of the median (right plot in Figure 8.20).

```
box_inf=boxplot(eclipse_rep$min.response.time, log="y",
                 boxwex=0.25, col="yellow", yaxt="n",
                 notch=TRUE, xlim=c(0.9, 1.3), ylab="")
```

The idea behind kernel density can be used to enhance boxplots to create what is known as a *violin plot*. The curve in Figure 8.21 is the kernel density of the data (a mirror image is usually included in the plot, hence the name). The beanplot function in the beanplot package supports a variety of functionality, including plotting the envelope of a violin plot.

```
library("beanplot")
beanplot(eclipse_rep$min.response.time, col="yellow", log="y",
         what=c(1, 1, 0, 0), ylab="Seconds\n")
```

The boxplot function supports multiple, related, boxplots in the same plot using formula notation, with the following code producing Figure 8.22 (portions of the notch are offset to avoid obscuring the hinge):

```
boxplot(time ~ group+task, data=gs, notch=TRUE, horizontal=TRUE,
        xlab="Time (minutes)")
```

A bar chart with error bars is regularly used to visually summarise values (sometimes known as *dynamite plots*). A study<sup>330</sup> investigating the effectiveness of various ways of visually summarizing data (including boxplots, violin plots and others) found that when extracting information from bar charts (with or without error bars) subjects did not perform as well as they did when using the other techniques.

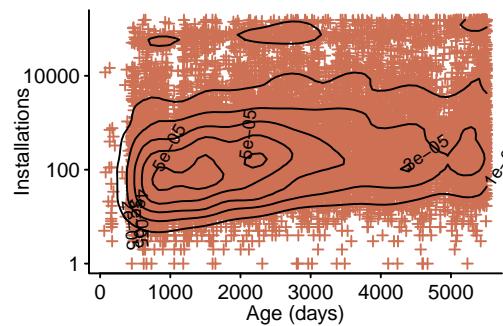


Figure 8.18: Number of installations of Debian packages against the age of the package; middle plot was created by smoothScatter and lower plot by contour. Data from the "wheezy" version of the Ultimate Debian Database project.<sup>1578</sup> [code](#)

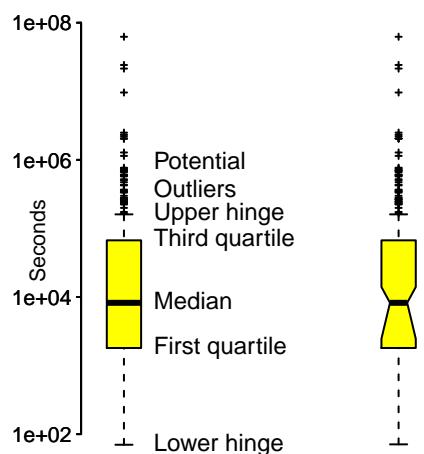


Figure 8.20: Boxplot of time between a potential mistake in Eclipse being reported and the first response to the report; right plot is notched. Data from Breu et al.<sup>207</sup> [code](#)

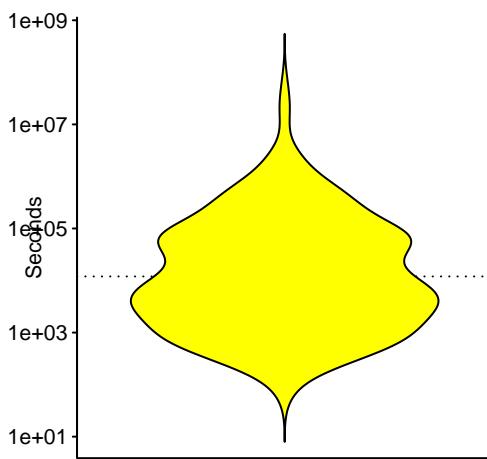


Figure 8.21: Violin plot of time between bug being reported in Eclipse and first response to the report. Data from Breu et al.<sup>207</sup> [code](#)

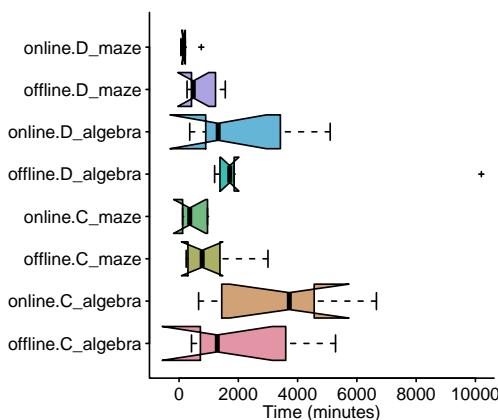


Figure 8.22: Time taken for developers to debug various programs using batch processing or online (i.e., time-sharing) systems. Data kindly provided by Prechelt.<sup>1277</sup>

### 8.2.6 Relationships between items

The relationship between two entities may be the attribute of interest. Graphs are the data structure commonly associated with relationships, and the `igraph` package contains numerous functions for processing graphs.

When displaying graphs containing large numbers of nodes, potentially useful information in the visual presentation may be swamped by many nodes having relatively few connections. Figure 8.23 is an attempt to show which languages commonly occur, within the same project, with another language, in a sample of 100,000 GitHub projects. The number of projects making use of a given pair of languages is represented using line width and to stop the plot being an amorphous blob the color and transparency of lines also changes with number of occurrences.

Perhaps items having relatively few connections are the ones of interest. The Microsoft Server protocol specifications<sup>1076</sup> contain over 16 thousand pages, across 130 documents (the client specification documents are also numerous). The upper plot in Figure 8.24 shows dependencies between the documents (based on cross-document references in the 2009 release<sup>1076</sup>); the lower plot shows the dependencies after excluding the 18 most referenced documents (plot based on the following code):

```
library("igraph")
library("sna")

+ interest_gr=graph.adjacency(interest, mode="directed")

# V(interest_gr)[names(in_deg)]$size=3+in_deg^0.7
V(interest_gr)$size=1
V(interest_gr)$label.color="red"; V(interest_gr)$label.cex=0.75
E(interest_gr)$arrow.size=0.2

--> plot(interest_gr)
```

It is possible to use R to draw presentable graphs, however, if your primary interest is drawing visually attractive graphs containing lots of information, then there other systems that may be easier to use (e.g., GraphViz<sup>615</sup>). Yes, an R interface to these systems may be available, but if statistical analysis is not the primary purpose, why is R being used?

Alluvial plots are a method for visualizing the flow between connected entities. Figure 8.25 shows factors used to prioritize the application of Github pull requests and the relative orders in which they appear in a dataset of pull requests;<sup>606</sup> the `alluvial` package was used.

## 8.2.7 3-dimensions

We live in a world of three spatial dimensions, which is only one more than the two dimensions available on flat screens and paper; various techniques for enhancing a flat surface to display information in one more dimension are available.

Heatmaps use color to display information about a third quantity within a 2-D plot. Figure 8.26 shows the L3 cache bandwidth (color+number) of an Intel Sandy Bridge processor running at various clock frequencies and using various combinations of cores.

Both the `heatmap` function in the base system and the `heatmap.2` function in the `gplots` package, clusters the rows/columns and then plots a dendrogram; various arguments have to be set to switch off this default behavior, with `heatmap` doing its best to make life difficult including not coexisting with other plots in the same image; `heatmap.2` is more reasonable.

The `levelplot` function in the `lattice` package provides straightforward functionality for producing heat maps, and it is used to produce all the heatmaps in this book.

```
library("lattice")  
  
t=levelplot(L3_band,  
            col.regions=rainbow(100, end=0.9),  
            xlab="Clock frequency (Mhz)", ylab="Cores used",  
            scales=list(x=list(cex=0.70, rot=35),  
                        y=list(cex=0.65)),
```

```
panel=function(...)  
{  
  panel.levelplot(...)  
  panel.text(1:11, rep(1:8, each=11),  
            L3_band, cex=0.55)  
}  
  
plot(t, panel.height=list(3.8, "cm"), panel.width=list(6.2, "cm"))
```

A contour plot can be used for visualizing the relationship between a response variable and two explanatory variables; the `contour` function is part of the base system. A study by Thereska, Doebel, Zheng and Nobel<sup>1548</sup> measured the performance of various applications running on a variety of desktop computers; the cpu speed and memory capacity of the computer hosting each of the 4,924,467 user sessions were recorded. The contours in Figure 8.27 are based on the number of user sessions measured on computers having a given processor speed and memory capacity.

```

library("plyr")

Um=unique(memcpu$MemorySize)
M_map=mapvalues(memcpu$MemorySize, from=Um, to=rank(Um))

Us=unique(memcpu$ProcSpeed)
S_map=mapvalues(memcpu$ProcSpeed, from=Us, to=rank(Us))

cnt_mat=matrix(data=0, nrow=length(Us), ncol=length(Um))

cnt_mat[cbind(S_map, M_map)]=log(memcpu$Session_Count)

contour(x=seq(min(Us)/max(Us), 1, length.out=length(Us))
        y=seq(min(Um)/max(Um), 1, length.out=length(Um))
        z=cnt_mat, col=pal_col, nlevels=10, axes=FALSE,
        xlim=c(min(Us)/max(Us), 1), ylim=c(min(Um)/max(Um), 1),
        xlab="Processor speed (GHz)",
        ylab="Memory size (Mbyte)\n")

axis(1, at=sort(Us)/max(Us), labels=sort(Us))
axis(2, at=sort(Um)/max(Um), labels=sort(Um))

```

A variety of functions are available for representing a 3-D plot as on 2-D surface, including the `scatterplot3d` function in the `car` and the `plot3d` function in the `rgl` package.

Histograms in 3-dimensions provide more opportunities, than histograms in 2-dimensions, for looking impressive with little data and misleading viewers. A study by Hamill and Goseva-Popstojanov<sup>654</sup> investigated the origin of 1,257 faults in 21 large safety critical applications, recording where the fixes were made (e.g., requirements, design, code or supporting files). Figure 8.28 shows a 3-D histogram of root cause/fix location on the x-y axis and a count of occurrences on the z-axis. Color has the effect of enhancing the visual appearance of the plot, and makes it easier to locate stacks having similar values, but it is very difficult to obtain detailed information from this plot. Adding numeric values would provide detail, but the real issue is what information is the plot intended to communicate?

```

library("lattice")
library("latticeExtra")

# log transform pulls out small differences in majority of counts
transform_breaks= exp(do.breaks(range(log(1e-4+STVR_col$occurrences)), 20))
t=cloud(occurrences ~ fix+fault, STVR_col,
         panel.3d.cloud=panel.3dbars,
         xlab="Fixes involved", ylab="Fault found", zlab="Count",
         xbase=0.5, ybase=0.5, aspect=c(1, 1),
         col.facet = level.colors(STVR_col$occurrences,
                                   at = transform_breaks,
                                   col.regions = rainbow),
         scales=list(arrows=FALSE, distance=c(2, 1.1, 1),
                     x=list(rot=-20) # Rotate tick labels
                    ))
plot(t)

```

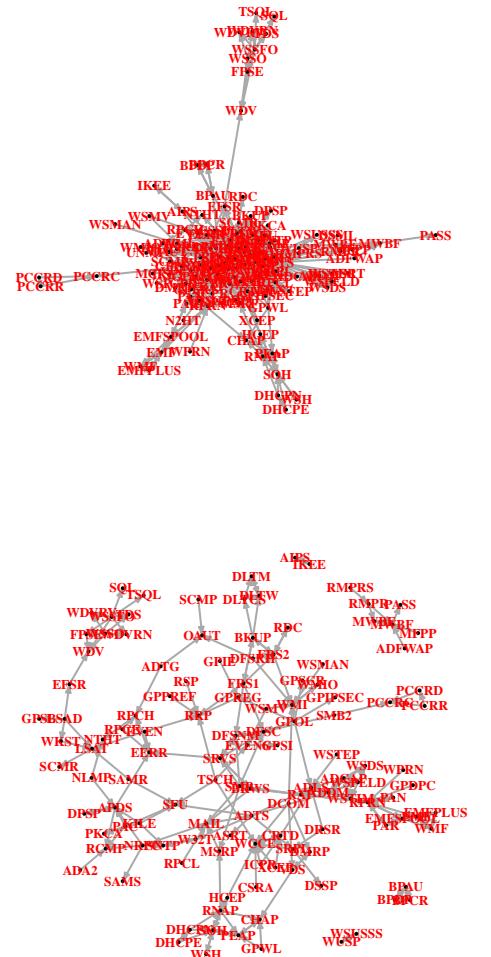


Figure 8.24: References from one document to another in the Microsoft Server Protocol specifications. Data extracted by the author from the 2009 document release.<sup>1076</sup>

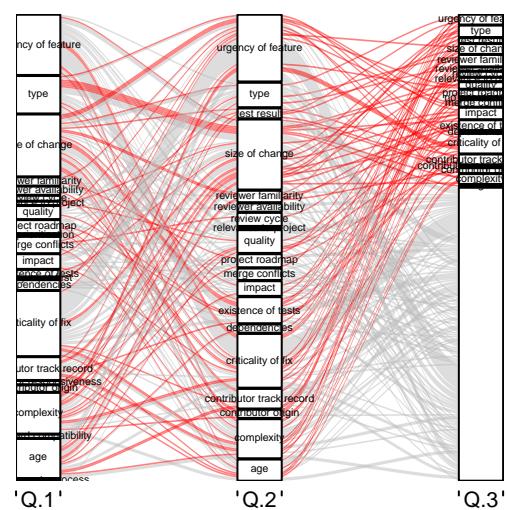


Figure 8.25: Alluvial plot of relative prioritization order of selection and application of Github pull requests. Data from Gousios et al.<sup>606</sup> [code](#)

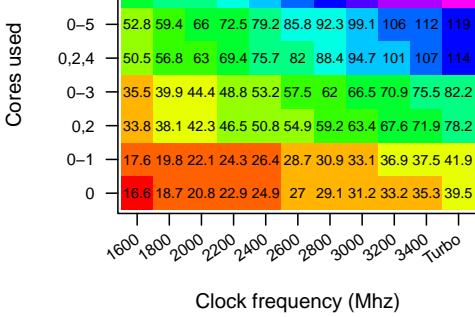


Figure 8.26: Intel Sandy Bridge L3 cache bandwidth in GB/s at various clock frequencies and using combinations of cores (0-3 denotes cores zero-through-three, 0,2,4 denotes the three cores: zero, two and four). Data from Schone et al.<sup>1389</sup> code

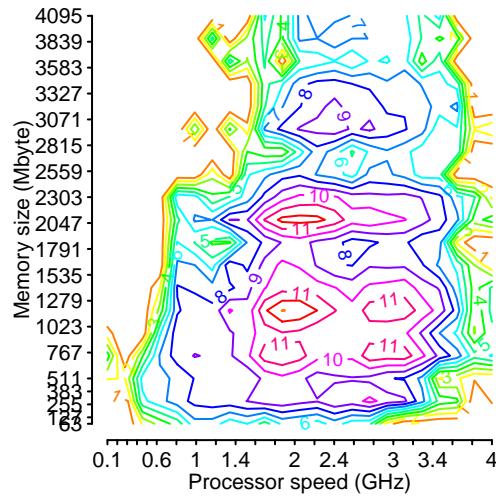


Figure 8.27: Contour plot of the number of sessions executed on a computer having a given processor speed and memory capacity. Data kindly provided by Thereska.<sup>1548</sup> code

A ternary, or triangle, plot has three axes. The axes are inclined at an angle of 60° to each other, and practice is needed to become proficient at estimating the coordinates of any point. Figure 8.29 shows two ways of labelling a ternary plot (with the three coordinates summing to 100%), with labels appearing at the vertex rather than along the axis and axis scales drawn either perpendicular to the axis or labeled along the axis and within the triangle as a grid. The upper plot shows how lines perpendicular to the appropriate axis are used to find the location of a point (at 10, 35, 55 in this case).

The closer points are to a vertex the larger the value of the corresponding variable, the closer points are to an axis the smaller the value of the corresponding variable.

Ternary plots are used to visualize compositional data (see Figure 5.33); the compositions and vcd packages include support for creating ternary plots.

In the following code rcomp normalises its argument (so that rows sum to 100) using an interval scale and returns an object having class rcomp (the compositions package has overloaded functions for handling objects of this type):

```
library("compositions")
xyz=c(10, 35, 55)
plot(rcomp(xyz), labels="", col="red", mp=NULL)
ternaryAxis(side=-1:-3, labels=paste(seq(20, 80, by=20), "%"),
            pos=c(0.5,0.5,0.5), col.axis=hcl_col, col.lab=pal_col,
            small=TRUE, aspanel=TRUE,
            Xlab="X", Ylab="Y", Zlab="Z")
lines(rcomp(rbind(xyz, c(10, 45, 45))), col=hcl_col[4])
lines(rcomp(rbind(xyz, c(32, 35, 33))), col=hcl_col[4])
lines(rcomp(rbind(xyz, c(22, 23, 55))), col=hcl_col[4])
plot(rcomp(xyz), labels="", col="red", mp=NULL)
isoPortionLines(col=hcl_col[4])
ternaryAxis(side=0, col.axis=hcl_col, small=TRUE, aspanel=TRUE,
            Xlab="X", Ylab="Y", Zlab="Z")
```

## 8.3 Communicating a story

Results from data analysis have no value until they have been reliably communicated to the target audience.<sup>295</sup> Reliably communicating information to other people is difficult; the intended message may be misunderstood or important parts may simply be overlooked by readers. The data analyst has to provide a narrative that tells the intended story.

No known algorithm is available that selects a method that ensures communication is made in a way that will be correctly interpreted by the audience. The main techniques available for presenting numeric information, and how they might be implemented using R, are covered in the rest of this chapter.

There are a wide variety of different ways of presenting information, e.g., tables, pie charts, bar charts and scatter plots. Which of these is best at communicating information to readers? The answer from a wide range of studies is that it depends on what information readers are trying to obtain. The following is a brief summary of some research findings:

- graph or table? Studies have found that except for reading-off specific values (and recall of these values later), subjects perform better with line graphs, than tables. However, while graphs have better performance when presenting a given perspective (e.g., by selection of the axis), tables may be preferable<sup>197</sup> when wanting to present data in a way that does not favour any one perspective on the data; it boils down to selecting the best cognitive fit,<sup>1604</sup>
- the ability of pie charts to communicate information has been questioned over the years.<sup>342</sup> A study<sup>1474</sup> comparing subject performance using pie charts, a horizontal divided bar chart, a vertical bar charts and a table, found that except when direct magnitude estimation was required pie charts were comparable to bar charts, but for combinations of proportions pie charts were superior; a study<sup>1452</sup> comparing the three visual clues present in a pie chart

(i.e., angle, area and circumference length) found that angle and area were poor methods of communicating information, and that circumference length was the best of three,

- adding a third dimension to a graph has been found to slow down reader performance,<sup>695</sup> i.e., subjects take longer to extract information and may be less accurate. The conclusion would appear to be, don't use three dimensions when two will do. While subjects have expressed a preference for using 3-D graphs to impress others, no studies have investigated whether they have this effect,
- studies<sup>7</sup> have found that human judgement of the relative sizes of surface areas has been found is based on a linear scale, e.g., a circle of radius two is likely to be around three times larger, rather than four times larger, than a circle of radius one.<sup>i</sup> Encoding information as an area can lead to reader communication breakdown; around a quarter of people focus on the areas of each slice in pie charts rather than either the length of the outer arc or the angle at the center of the chart (which have a linear correlation with the information intended to be communicated),
- studies by Cleveland are often cited in R related publications: one study<sup>304</sup> asked subjects to make judgements about graphical information encoded in various ways<sup>ii</sup>; the results showed that accuracy of subjects' answers varied slightly between encoding methods, the ordering from most accurate to least accurate was: position along a common scale, positions along nonaligned scales, length, direction, angle, area, volume, curvature and shading, color saturation. Later studies<sup>1474</sup> suggest that the factors are not so well-defined, with some effects found to be influenced by the structure of the experiments, or performance with a particular encoding depending on the task subjects' performed.

The thinking behind some layout details used by R's plot function are based on experimental work by Cleveland.<sup>303</sup> Although not explicitly stated the aim appears to have been to present data in a workman-like way that avoids the possibility of plotted data values being obscured by plot markings (e.g., tick marks).

The plot function is a workhorse for handing the graphical display of data in R; it does a good job of producing a reasonable looking plot from whatever it is passed. Based on this book's implementation goal of using one implementation technique, where-ever possible, the plots in this book were generated using the plot function.

The lattice<sup>136</sup> and ggplot<sup>269</sup> packages provide alternative world views on the plotting of data; lattice is based on the Trellis graphics system<sup>128</sup> from Bell Labs and has an emphasis on multivariate data, while the design of ggplot is derived from the work of Wilkinson.<sup>1668iii</sup> Both lattice and ggplot provide a great deal of control over the created plot through the use of user supplied functions. While ggplot is widely used by experienced R developers, its inability to sensibly handle whatever nonsense data is thrown at it (e.g., nonsense in that there are mistakes in the R code that produced it) prevents this package being recommended for casual use. A detailed technical overview of the R graphics subsystems is available in "R Graphics" by Murrell.<sup>1122</sup>

Combining data with visual information familiar to readers helps them to extract patterns that mean something to them. Figure 8.30 shows single event upsets (i.e., radiation induced memory faults) experienced by NASA's Orbview-2 spacecraft during one day in 2000. Overlaying the satellite location at the time of the upset on a map of the Earth (using the map package) provides context to help readers understand where most upsets occur.

In some cases the intent of a plot may be to communicate that life is complicated. For instance, Figure 8.31 shows an estimate of the market share of Android devices in use in 2015, by brand/company and product name (based on the 682,000 unique devices that downloaded an App from OpenSignal<sup>1193</sup>).

```
library("treemap")
and_tree=treemap(android, c("brand", "model"), "august2015",
                 title="", palette=pal_col,
                 border.col="white", border.lwds=c(0.5, 0.25))
```

<sup>i</sup> Stevens' power law states that psychological magnitude is proportional to the actual magnitude raised to some power. Unlike many other quantities used to express information graphically, for area this power is 0.8 rather than 1 (although values close to 1 have been found for some tasks).

<sup>ii</sup> Replication.<sup>675</sup>

<sup>iii</sup> The title of Wilkinson's book "The Grammar of Graphics" refers to the structure of software written to display graphics rather than the structure of the displayed information.

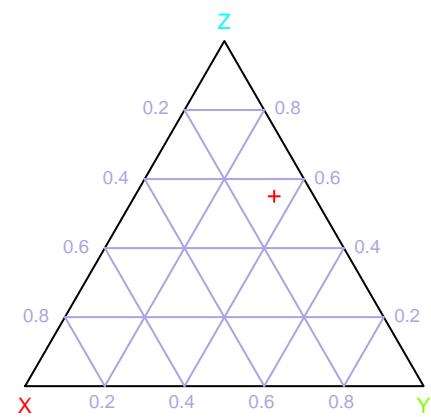
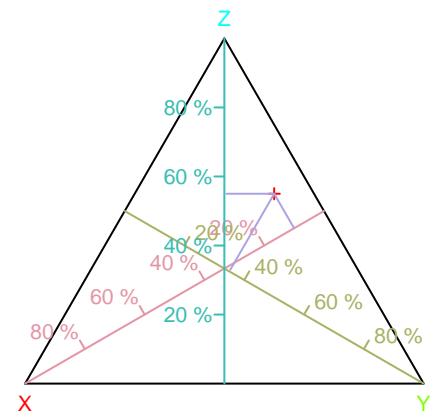


Figure 8.29: Ternary plots drawn with two possible visual aids for estimating the position of a point (red plus at  $x=0.1$ ,  $y=0.35$ ,  $z=0.55$ ); axis names appear on the vertex opposite to the axis they denote. [code](#)

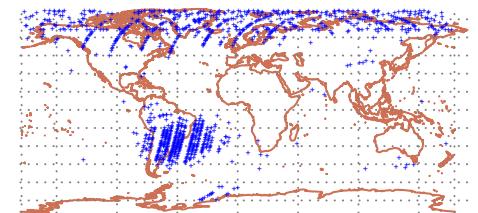


Figure 8.30: Earth relative positions of NASA's Orbview-2 spacecraft when it experienced a single event upset (in blue) on 12 July 2000. Data kindly provided by LaBel. [code](#)

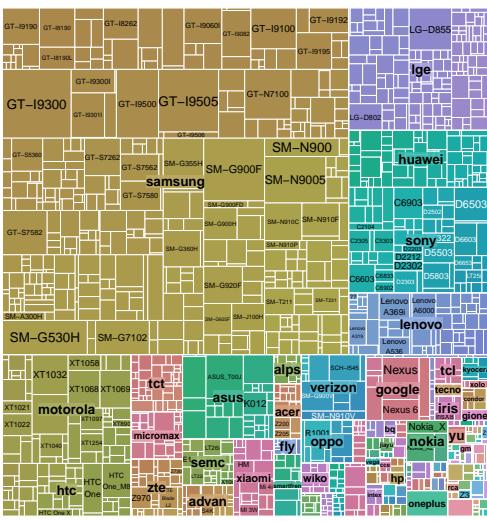


Figure 8.31: Estimated market share of Android devices by brand and product, based on downloads from 682,000 unique devices in 2015. Data from OpenSignal.<sup>193</sup> [code](#)

### 8.3.1 What kind of story?

The kinds of output from statistical data analysis include the following:

- a description of the data, e.g., its mean and variance, how measurements cluster, an equation summarizing the data. A descriptive model, built from the data, can be used to help gain insights into the system that was measured, for comparing different descriptions (e.g., benchmark results) and for building similar systems (e.g., automatically creating file system contents<sup>14</sup> for benchmarking purposes)
  - a model built to mimic the behavior of a system (as expressed in the measurements made), e.g., a simulator,
  - a predictive model capable of making appropriately accurate predictions for values not in the set of measurements used to build the model. The possible range of prediction values may be within the range of values used to build the model or outside the range of these values, e.g., making predictions about a future time,

A standard reply to any complaints about the adequacy of a model built using data is the adage “All models are wrong but some are useful.”

An example of the different kinds of model that can be built, and how their usefulness depends on the problem they are intended to solve, is provided by a question involving the use of local variables in the source code of a function definition.

If the source code contains a total of  $N$  read accesses to variables defined locally within the function, what percentage of variables will be read from once, twice and so on (based on a static count of the visible source code, not a dynamic count obtained by executing the function)?

Using data from an analysis of C source<sup>785</sup> we have a description of "what is".

Plotting the data shows that a few variables account for most of the accesses (i.e., read from). After some experimentation the following equation was found to be a good fit to the data; see Figure 8.32:

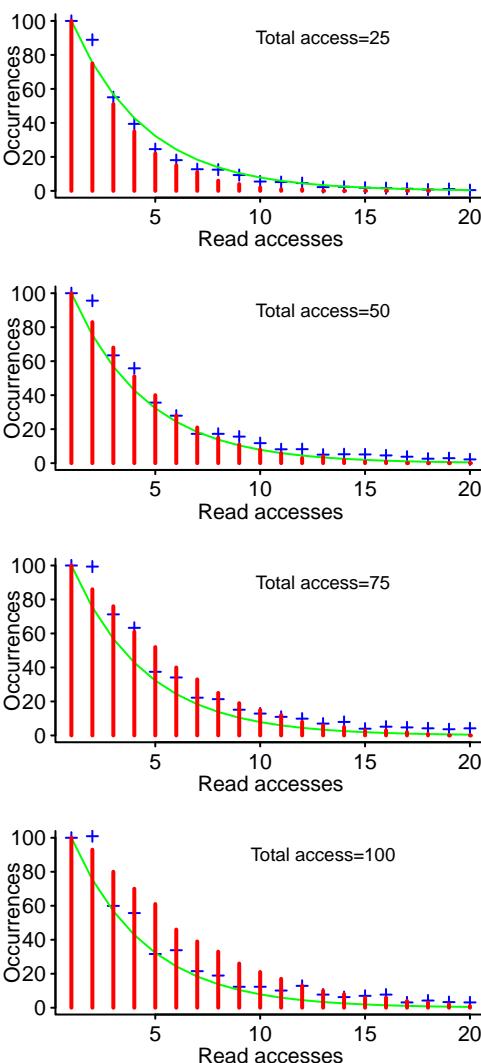


Figure 8.32: Variables having a given number of read accesses, given 25, 50, 75 and 100 total accesses, calculated from running the weighted preferential attachment algorithm (red), the smoothed data (blue) and a fitted exponential (green). [code](#)

What other kind of model can be built to answer this question?

This problem has a form that has parallels with the growth of new pages and links to existing pages on the world wide web. Each access of a local variable could be thought of as a link to the information contained in that variable. One idea that has been found to be integral to modeling the number of links between web pages is *Preferential attachment*.

With some experimentation an iterative algorithm based on preferential attachment, was created, that produced a pattern of behavior close to that seen in the data. The algorithm is as follows:

Assume we are automatically generating code for a function, and from the start of the function to the current point in the code  $L$  distinct local variables exist (and have been accessed), with each accessed  $R_i$  times ( $i = 1, \dots, L$ ). The following weighted preferential attachment algorithm is used to select the next local variable to access (global variables are ignored in this analysis):

- With probability  $\frac{1}{1+0.5L}$  create a new variable to access, <sup>iv</sup>
  - with probability  $1 - \frac{1}{1+0.5L}$  select a variable that has previously been accessed in the function, choose an existing variable with probability proportional to  $R + 0.5L$  (where  $R$  is the number of times the variable has previously been read from); e.g., if the total accesses up to this point in the code is 12, a variable that has had four previous read accesses is  $\frac{4+0.5 \cdot 12}{2+0.5 \cdot 12} = \frac{10}{8}$  times as likely to be chosen as one that has had two previous accesses.

---

<sup>iv</sup> The unweighted preferential attachment algorithm uses a fixed probability to decide whether to access a new variable.

The red points in Figure 8.32 were calculated using the above algorithm.

This preferential attachment model provides insights into local variable usage that are very different from those provided by the fitted exponential equation. Neither of them could not be said to be realistic descriptions of the process used by developers, when writing code. Both models are descriptions of the end result of the emergent process of writing a function definition; each model has its own advantages and disadvantages, including the following:

- the fitted equation is fast and simple to calculate, while the output from the iterative model is slow (an average over 1,000 runs in the example code) and requires more work to implement,
- the iterative model automatically generates a possible sequence of accesses (for machine generated source), while a fitted equation does not provide any obvious method of generating a sequence of accesses,
- multiple executions of an iterative model can be used to obtain an estimate of standard deviation, while the equation does not provide a method for estimating this quantity (it may be possible to fit another regression model that provides this information),
- the equation provides an end result way of thinking, while the iterative model provides a choice-based way of thinking about variable usage.

A common technique for devising a model for a new problem is to find a very similar problem that has a proven model, and to adapt this existing model to the new problem. A model based on existing practice is often easier to sell to an audience, than a completely new model.

Some multiprocessor system have a "shared nothing" architecture, which minimises the sharing of hardware resources. Performance measurements of such systems, under various loads, shows that even when tasks can be evenly distributed across all  $X$  processors in the system, performance is rarely  $X$  times faster. Which model provides a good explanation of the performance seen?

*Amdahl's law* predicts changes in multiprocessor performance as the number of processors used changes, where the multiprocessor system has a shared hardware architecture. Gunther<sup>641</sup> extended this "law" to cover multiprocessors having "shared nothing" architecture; the adapted model, plus a further adaption, are not good fits to the data (see Figure 8.33).

Gunther<sup>642</sup> subsequently created a model based on queuing theory and ran simulations to model performance (with each job waiting in a queue for time  $t_1$  and executing for time  $t_2$ ). The argument for using queuing theory was that data sharing between different programs can create a resource contention that the "shared nothing" hardware architecture cannot unblock. Figure 8.33 shows that the queuing model more accurately follows the pattern measured. Given the small amount of data available it would be unwise to attempt further model tuning.

The R language does not contain features designed with simulation in mind<sup>v</sup>, but like most languages it can be adapted to solve problems outside of its core domain; see the `simFrame` and `simmer` packages.

Finding a workable model, based on the available data, can involve many iterations over a long time. For instance, modeling the growth of the size and number of files/directories in a filesystem has a long history, with current models<sup>1096</sup> either involving a mixture of two distributions for the equation fitting approach or a generative approach based on simulating the way new files are created from existing files.

Perhaps the most important question to answer when proposing any model is the purpose to which it will be put. A model intended to gain insight might not be of any use in making practical recommendations and a model used to make predictions might not provide any useful insight. For instance, modeling the connection between modifications to files and the introduction of mistakes which cause faults to be experienced may be used to predict coding mistake rates based on modification history, but such a model has limited scope for directly deriving methods for reducing faults (e.g., reduce faults by reducing file modifications, is of no use when customers want new or modified behavior in the applications they use).

In most cases, a great deal of domain knowledge is required to build a model having the desired level of performance. There is no guarantee that any created model will be sufficiently

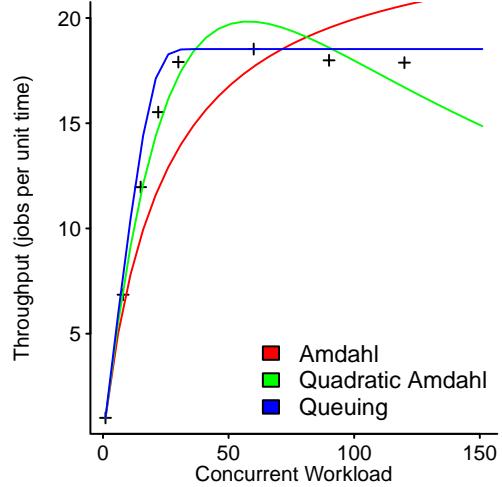


Figure 8.33: Throughput when running the SPEC SDM91 benchmark on a Sun SPARCcenter 2000 containing 8 CPUs, with the predictions from three fitted queuing models. Data from Gunther.<sup>642</sup> [code](#)

<sup>v</sup> Interfacing to NetLogo<sup>1282</sup>

accurate to be useful for the problem at hand; this is a risk that occurs in all model building exercises. Ideally model building is driven by a theory describing the behavior of the system being modeled. When a theory is complete there is no need for new models, the fact that the creation of a new model is being considered implies that existing models are lacking in some respect.

### 8.3.2 Technicalities should go unnoticed

The machinery of information presentation should not get in the way of reader's access to that information.

There are many books offering tips, suggests and recommendations for how best to present visual information; the only book recommended by your author (it is based on a wide range of empirical research) is "Graph Design for the Eye and Mind" by Stephen Kosslyn.<sup>881</sup> Sometimes multiple plots are used to tell an evolving story, McCloud<sup>1036</sup> is a great introduction to this art form.

#### 8.3.2.1 People have color vision

Until the mid 1980s most people used computer terminals that were only capable of displaying black and white (or green and black). Forty years later, the look-and-feel of mid-1980s computer usage still predominates in serious works of data visualization.

This book treats color as an essential component of numeric story telling. Color provides an extra dimension that enables more information to be present within the same area, and make it easier for viewers to extract information from a plot.<sup>vi</sup>

Selecting the most appropriate colors to use requires skill and experience. The `colorspace` and `RColorBrewer` packages both include functions that automatically select a color palette based on the arguments passed;<sup>vii</sup> the `colorspace` package provides a wider range of functionality than `RColorBrewer`, and is used to select the colors for the plots appearing in this book.

The Hue-Chroma-Luminance (HCL) color space is claimed<sup>1696</sup> to provide a better mapping to the human color perceptual system (hue: dominant wavelength; chroma: colorfulness, intensity compared to gray; and luminance: brightness, amount of gray), than alternative spaces.<sup>viii</sup> The color palettes generated by the `rainbow_hcl` function are considered to be qualitative palettes, that are suitable for depicting different categories; those generated by the `sequential_hcl` function to be suitable for coding numerical information that ranges over a given interval, with the `diverge_hcl` function also encoding numerical information, but including a neutral value.

The `choose_palette` function provides an interactive, slider based, method for developers to define their own color palettes.

Approximately 10% of men and 1% of women have some form of color blindness. The `dichromat` package provides a way of showing how a plot containing color would appear to a viewer having some form of color blindness. The package makes use of experimental data<sup>1612</sup> to simulate the effects of different kinds of color blindness, modifying the requested colors to appear, to normal sighted viewers, like they would to a viewer having the selected kind of color blindness.

#### 8.3.2.2 Color palette selection

Figure 8.36 shows how time varying data involving related items (in this case market share of successive versions of Android) can be displayed in a way that preferentially highlights one aspect of the data; the upper plots highlighting individual versions while the lower plots show each version's contribution to the overall market share. Bold colors are effective at drawing

<sup>vi</sup> R contains 657 built-in color names (the `colors` function lists them) and also supports hexadecimal RGB literals.

<sup>vii</sup> The selection process is based on theories derived from the use of color in maps,<sup>208</sup> which has a long history.

<sup>viii</sup> Red-Green-Blue (RGB) is a specification based on the display of color on computer screens; Hue-Saturation-Value (HSV) is a transformation of RGB that attempts to map to the human perceptual system and is used by some other software packages.

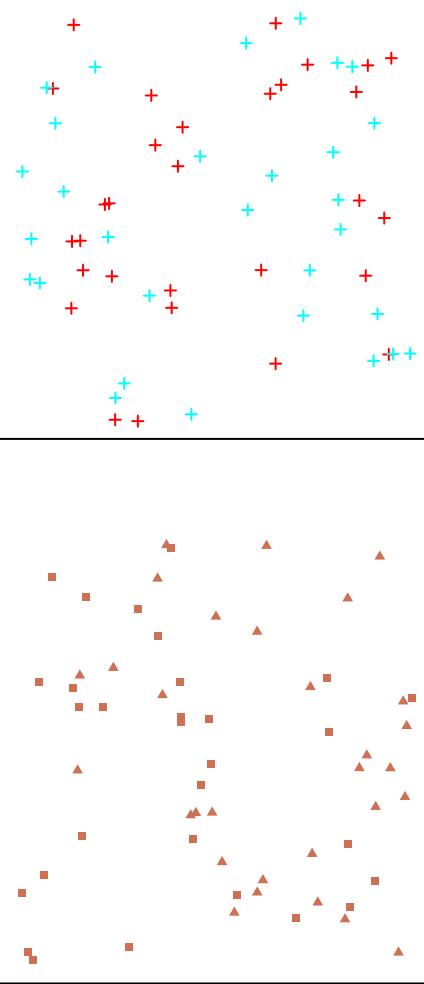


Figure 8.34: Illustration of the difference in cognitive effort needed to locate points differing by shape or color. `code`

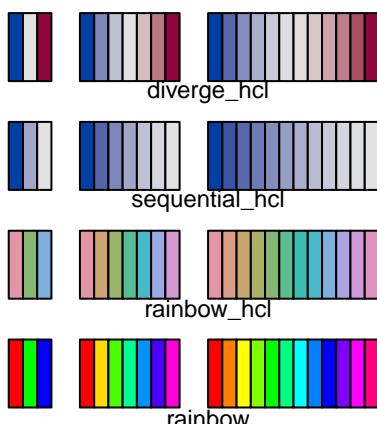


Figure 8.35: The three, seven and twelve color palettes returned by calls to the `diverge_hcl`, `sequential_hcl`, `rainbow_hcl` and `rainbow` functions. `code`

attention to individual lines, but can be overpowering when there is a large area of color in the plot; the opposite is often the case for pastel colors.

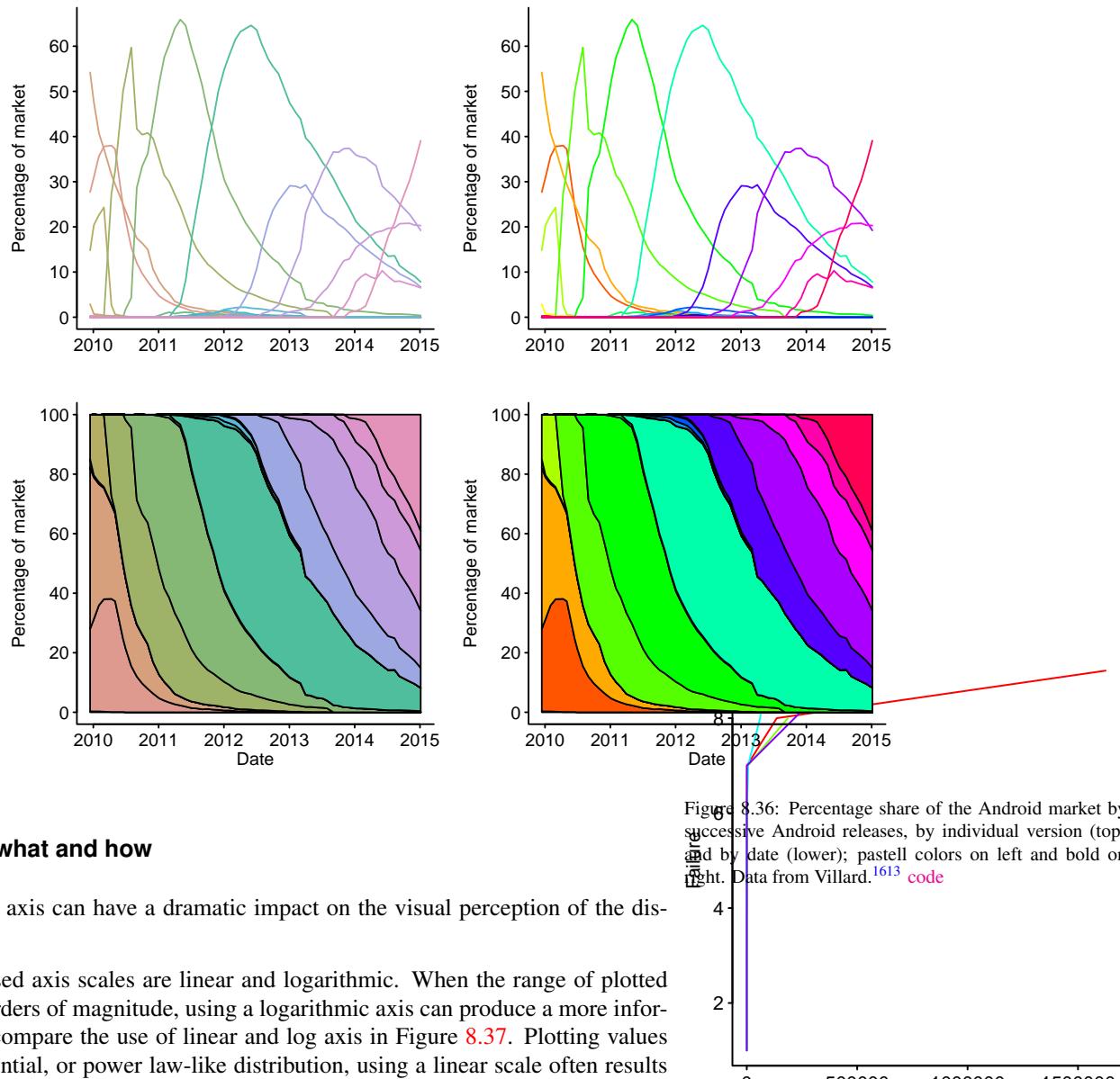


Figure 8.36: Percentage share of the Android market by successive Android releases, by individual version (top) and by date (lower); pastel colors on left and bold on right. Data from Villard.<sup>1613</sup> code

### 8.3.2.3 Plot axis: what and how

The choice of plotting axis can have a dramatic impact on the visual perception of the displayed data.

The two commonly used axis scales are linear and logarithmic. When the range of plotted values spans several orders of magnitude, using a logarithmic axis can produce a more informative visualization; compare the use of linear and log axis in Figure 8.37. Plotting values drawn from an exponential, or power law-like distribution, using a linear scale often results in many of the points being visually clumped together in a small area of the plot; use of a log scaled axis has the effect of spreading out these clumped values.

The plot function (and many other plotting functions) automatically select the minimum-/maximum range of each axis, based on the range of the data passed; by default 4% is added to each end of the range.

The choice of quantity plotted along each axis is driven by the relationship, between two quantities, that the analyst is seeking to highlight; the purpose of a plot is to communicate this relationship.

Care needs to be taken to ensure that artificial relationships are not generated by the choice of quantity used one axis. An example of the wasted effort that can occur, when the relationship implied by the quantities plotted along an axis are not carefully analysed, is provided by the saga of fault density vs. lines of code.

It was noticed that when fault density (i.e., number of faults divided by lines of code in a function) was plotted against lines of code (in a function), the distribution of points had a pattern that resembled a lopsided U. Some researchers proposed that the minimum of this U represented an optimum for the length of a function.<sup>661</sup>

A study by El Emam, Benlarbi, Goel, Melo, Lounis and Rai<sup>438</sup> showed that this U-shape was an artefact generated by the choice of quantities plotted along each axis. Plotting the ratio  $\frac{F}{LOC}$  against  $LOC$ , with  $F$  constant, will produce a tilted U-shape (blue line in Figure 8.38). If the number of faults grows faster than the number of lines of code (which has been found

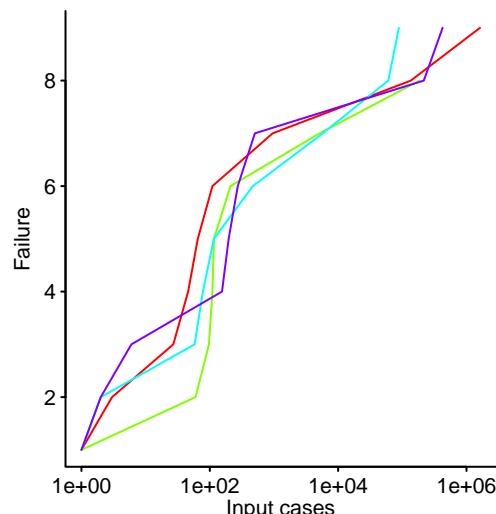


Figure 8.37: Input case on which a failure occurred, for a total of 500,000 inputs; values plotted using a linear (upper) and logarithmic (lower) x-axis. Data from Dunham et al.<sup>419</sup> code

to occur for large line counts) then U-shaped curves such as the red line in Figure 8.38 can occur (a growth rate was picked to illustrate one possibility).

```
x=1:100 ; inv.x=1/x

plot(x, 3*inv.x, type="l", col=pal_col[1],
      xlab="LOC", ylab="Faults/LOC\n")
lines(x, ((x+50)^3/5e4)*inv.x, col=pal_col[2])
```

The idea suggested by the U-shaped pattern, that there might be an optimal function length, is the result of misinterpreting the mathematical behavior of a ratio quantity plotted against one of the values used in the ratio calculation. The pattern seen in plots is an artefact of the choice of quantities chosen for each axis.

A log transform of an axis can sometimes hide potentially useful information, rather than help reveal it. Figure 8.39 is from a study by Putnam and Myers<sup>1290</sup> (their Figure 8.3); in both cases the x-axis is log transformed. In the right plot the y-axis is linear and there is a visually distinct cluster of measurements, across the top; in the lower plot, where both axes are log transformed, this cluster is visually less prominent.

Figure 8.38: Illustration of U-shape created when y-axis values are a ratio calculated from x-axis values. [code](#)

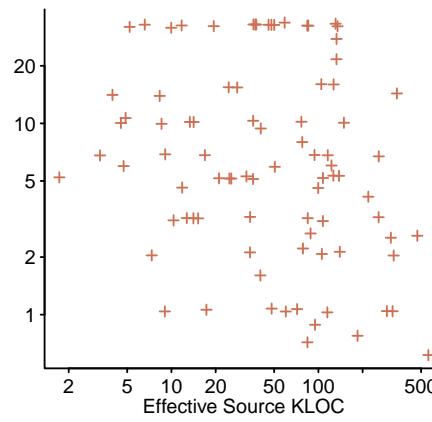
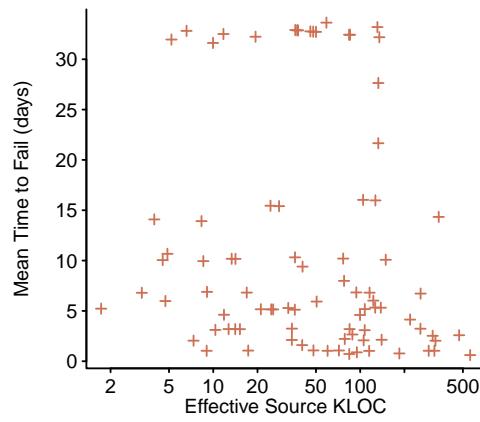
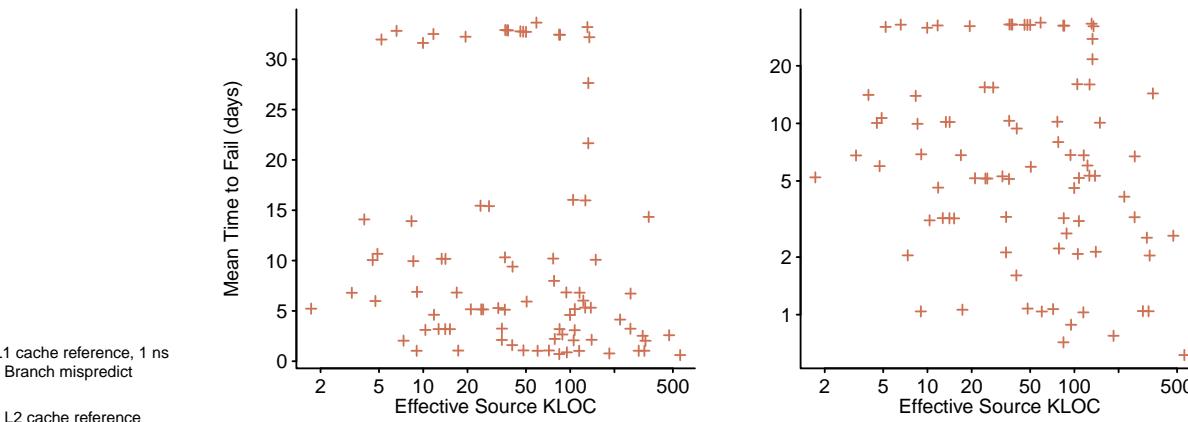
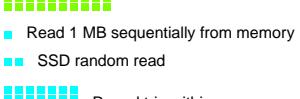
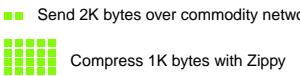
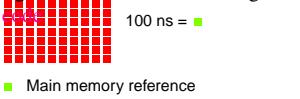


Figure 8.39: Mean time to fail for systems of various sizes (measured in lines of code); linear y-axis left, log y-axis right. Data extracted from Figure 8.3 of Putnam et al.<sup>1290</sup>



### 8.3.3 Communicating numeric values

The output from statistical analysis can include visual plots and a small collection of numbers. What is the best way to communicate a story involving a small collection of numbers?

The uncertainty associated with using descriptive phrases to denote probabilities is discussed in Section 2.7.1. Confidence intervals are a practical means of communicating uncertainty and are covered in Section 11.2.1.

A table of numbers covering a wide range of values (e.g., Table 8.2) can be difficult to interpret quickly unless this is something readers regularly do. An alternative representation separates out the mantissa and exponent, and combines them using area and color, allowing a visual same/different comparison to be made (see Figure 8.40).

Regression modeling (Section 10.5.1) finds a best fit of an equation to data, according to some specified definition of the error between the equation and data. While the numeric values (often referred to as *parameters*) are the output of model building, the information being communicated is equation+parameter values, i.e., the final fitted equation should be shown.

Packages are available for integrating the output from R programs into the workflow of various document preparation systems, for instance, the `ascii` package provides functions for producing Asciidoc compatible output and the `knitr` package produces LaTeX output.

Figure 8.40: Alternative representation of numeric values in Table 8.2. Data from Scott.<sup>1398</sup> [code](#)

Complicated equations can have unexpected behavior. Figure 8.41 shows the result of plotting the following set of equations:

Operation	Approximate runtime
L1 cache reference	1 ns
Branch mispredict	3 ns
L2 cache reference	4 ns
Mutex lock/unlock	17 ns
Main memory reference	100 ns
Send 2K bytes over commodity network	177 ns
Compress 1K bytes with Zippy	2,000 ns
Read 1 MB sequentially: memory	7,000 ns
SSD random read	16,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially: magnetic disk	1,000,000 ns
Seek: magnetic disk	3,000,000 ns
Send packet CA→Netherlands→CA	150,000,000 ns

Table 8.2: Numbers Everyone Should Know, circa 2016. Data from Scott.<sup>1398</sup>

$$-4.7 \leq x \leq 4.7$$

$$y_1 = c(1, -0.7, 0.5) \sqrt{c(1.3, 2, 0.3)^2 - x^2} - c(0.6, 1.5, 1.75)$$

$$y_2 = \frac{0.6\sqrt{4-x^2}-1.5}{1.3 \leq |x|}$$

$$y_3 = c(1, -1, 1, -1, -1) \sqrt{c(0.4, 0.4, 0.1, 0.1, 0.8)^2 - (|x| - c(0.5, 0.5, 0.4, 0.4, 0.3))^2} - c(0.6, 0.6, 0.6, 0.6, 1.5)$$

$$y_4 = \frac{c(0.5, 0.5, 1, 0.75) \tan\left(\frac{\pi}{c(4.5, 4.5)} (|x| - c(1.2, 3, 1.2, 3))\right) + c(-0.1, 3.05, 0, 2.6)}{c(1.2, 0.8, 1.2, 1) \leq |x| \leq c(3, 3, 2.7, 2.7)}$$

$$y_5 = \frac{1.5\sqrt{x^2+0.04}+x^2-2.4}{|x| \leq 0.3}$$

$$y_6 = \frac{2||x|-0.1|+2||x|-0.3|-3.1}{|x| \leq 0.4}$$

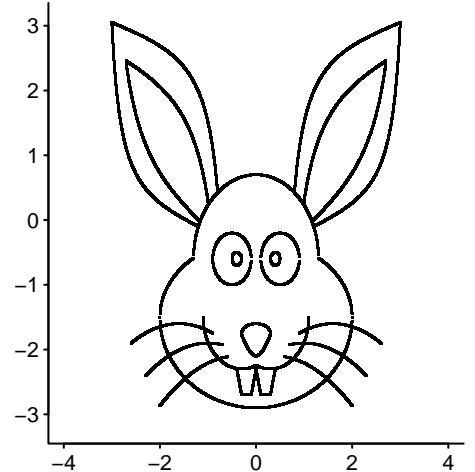
$$y_7 = \frac{-0.3(|x| - c(1.6, 1, 0.4))^2 - c(1.6, 1.9, 2.1)}{c(0.9, 0.7, 0.6) \leq |x| \leq c(2.6, 2.3, 2)}$$

Ease of generating fake data and modifying existing data... [communicating/warp-pts.R](#)

### 8.3.3.1 Percentages vs. frequencies

Studies have found that people are much better at extracting certain kinds of information when it is presented in the form of frequency of occurrence rather than as a percentage<sup>568</sup> ...

inline charts...

Figure 8.41: What's up doc? Perhaps, not the expected pattern in the data. Equations from White.<sup>1654</sup> [code](#)



# Chapter 9

# Probability

## 9.1 Introduction

What are the chances of an event occurring?

Probability is the mathematics involved in answering this question, and reasons for being interested in the estimation of probabilities include:

- betting, does a particular case need to be handled, making an insurance decision and for all other decisions and predictions,
- deciding the extent to which an event is surprising. The level of surprise might be used to decide whether something is going wrong or, when performing statistical analysis, discriminating between hypotheses.

Readers are assumed to have some basic notion of the concept of probability, and to have encountered the idea of probability in the form of likelihood of an event occurring; classic examples involve calculating the probability of a given combination or sequence of values occurring when flipping a coin or rolling a die, e.g., two heads or rolling two sixes, or the probability of having to make  $N$  flips/rolls before some event occurs.

What is the difference between probability and statistics?

Probability makes inferences about individual events based on the characteristics of the population, while statistics makes inferences about the population based on the characteristics of a sample of the population<sup>i</sup>.

Another way to compare the two is that probability makes use of deductive reasoning, while statistics makes use of inferential reasoning.

Probability and statistics are intertwined in that ideas and techniques from probability, about individual events, may be used when solving problems involving statistics and results about the characteristics of a population, obtained from statistical analysis, may be used to help solve problems involving probability.

People make use of various phrases to express their view of the likelihood of an event occurring (e.g., "almost impossible" and "quite possible"). Studies have found large cultural and personal differences in the numeric probabilities assigned to such phrases; see Figure 6.4 and Figure 2.54.

This book is data driven, and so primarily makes use of statistical analysis. The following example is a problem for which possible answers can be suggested using a probability model (data on developer behavior would provide evidence).

The vendor of a static analysis tool wants to add support for detecting a newly discovered pattern of mistakes made by developers. An occurrence of this pattern, in code, is not always a mistake. What is the upper bound on the probability of generating a false positive, that keeps the likelihood of developers continuing to use the tool above some limit (say 90%)?<sup>ii</sup>

<sup>i</sup> Statistics could be defined as the study of algorithms for data analysis.

<sup>ii</sup> Experience shows that tool false-positives are sufficiently unpopular (they are a source of wasted effort), that a developer will stop using the tool concerned if they are encountered too often. Higher false-positive rates for Tornado warnings result in more deaths and injuries, <sup>1443</sup> through people ignoring the warning.

Answering this question requires knowledge of the mental model used by developers to evaluate analysis tool performance. The following are two possible mental models (both assume zero correlation between difference warning occurrences and that developers assign the same importance to all warning messages):

- an *economic* developer who tracks the benefit of processing each warning (e.g., false positive warning  $-1$  benefit, else  $+1$  benefit), starting in an initial state of zero benefit this economic developer stops processing warnings if the current sum of benefits ever goes negative.

The Ballot theorem gives the probability that, when sequentially processing warnings, the number of true warnings is always greater than the number of false positive warnings (assuming equal weight is given to both cases, the alternative being more complex to analyse). Let  $C$  be the number of correct warnings and  $F$  the number of false positive warnings and assume  $C > F$ , then the probability is given by:

$$\frac{C - F}{C + F}$$

rewriting in terms of the probability of the two kinds of warning (i.e.,  $C + F = 1$ ), we get:

$$C_p - F_p$$

so, for instance, when the false positive rate is 0.25 the probability of a developer processing all the warning generated by a tool is  $0.75 - 0.25 \rightarrow 0.5$ , and does not depend on the total number of warnings.

- an *instant gratification* developer who processes each warning and stops when a sequence of  $N$  consecutive false positive warnings have been encountered. This kind of thinking is analogous to that of the *hot hand in sports* (what psychologists call the clustering illusion).

What is the probability that a sequence of  $N$  consecutive false positive warnings is not encountered?

If the total number of warnings is  $k$  and  $q$  is the probability of a false positive occurring, then the probability of a run of  $N$  consecutive false positive warnings occurring can be calculated using the following recurrence:

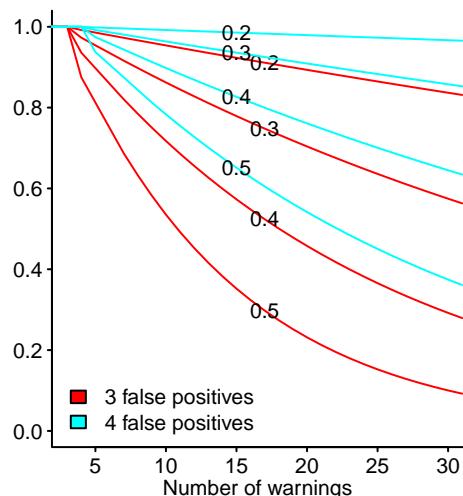
$$P(k, q, N) = P(k - 1, q, N) + q^N(1 - q)(1 - P(k - N - 1, q, N))$$

with initial values:

$$P(j, q, N) = 0, \text{ for } j = 0, 1, \dots, N - 1$$

$$P(j, q, N) = q^N, \text{ for } j = N$$

Figure 9.1 shows the probability of not encountering a sequence of three (red) or four (blue) consecutive false positive warnings when processing some total number of warning messages, for various underlying false positive rates (ranging from 0.5 to 0.2).



When dealing with warnings involving complex constructs, a developer may be unwilling to put the effort into understanding the situation and either goes along with what the static analysis tool reports, thus underestimating the actual false positive rate, or defaults to assuming the warning is a false positive, thus overestimating the actual false positive rate.

Finding an equation, or technique, to use in solving a problem involving probability requires some knowledge of the terminology used in this field. Possible phrases to try in search queries include: birth and death process, coin tossing, colored balls, combination, ergodic, event, fair games, first passage time, generating function, Markov chain, Markov process, occupancy problem, partitions, permutation, random walk, stochastic, trials and urn model.

Finding a closed form solution to an equation can be difficult, even when one exists. Sometimes the processes being studied contains so many interacting components that it is not possible to model them analytically; an alternative approach is simulation, discussed in Section 12.5.

### 9.1.1 Useful rules of thumb

If the distribution of the values taken by some attribute, in a population, is not known, the following inequalities can be used as worst case estimates of the probability of various relationships being true. Both inequalities are distribution independent (the price of this generality is that the bounds are loose).

### Markov inequality

The Markov inequality uses the sample mean,  $\mu$ , to calculate the maximum probability that  $X$  (which is required to be nonnegative) is larger than some constant. The inequality does not make any assumptions about the sample distribution:

$$P(X \geq k) \leq \frac{\mu}{k}$$

where  $\mu$  is the sample mean.

Example. If a sample has  $\mu = 10$ , then the probability of the sample containing a value greater than or equal to 20 (i.e., twice the mean) is:  $\frac{10}{20}$ .

### Chebychev's inequality

If the standard deviation,  $\sigma$ , of a sample is known, then Chebychev's inequality can be used to calculate a tighter bound than that given by the Markov inequality, as follows:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

alternatively:

$$P(|X - \mu| \geq k) \leq \frac{\sigma}{k^2}$$

Using the above example the probability of the sample containing a value that differs from the mean by at least 10 is less than or equal to:  $\frac{\sigma}{10^2}$ .

Example: an analysis of the number of mutants needed to estimate test suite adequacy to within a specified error and confidence bounds.<sup>597</sup>

### Fréchet inequalities

Bounds on the union and disjunction of two or more probabilities are given by the Fréchet inequalities, as follows:

Logical conjunction:  $\max(0, P(a_1) + P(a_2) - 1) \leq P(a_1 \wedge a_2) \leq \min(P(a_1), P(a_2))$

Logical disjunction:  $\max(P(a_1), P(a_2)) \leq P(a_1 \vee a_2) \leq \min(1, P(a_1) + P(a_2))$

**Correlation between three variable pairs** If the correlation between two pairs of three variables is known, say  $r_{12}$  and  $r_{13}$ , the bounds on the correlation of the remaining pair  $r_{23}$  is given by:

$$r_{12}r_{13} - \sqrt{(1 - r_{12}^2)(1 - r_{13}^2)} \leq r_{23} \leq r_{12}r_{13} + \sqrt{(1 - r_{12}^2)(1 - r_{13}^2)}$$

As the number of variables involved increases, the expressions become more complicated.<sup>225</sup>

### Rule of three

Say  $N$  colored balls are drawn from a box, and the number of balls of each color counted. If there are  $r$  red balls, a reasonable estimate of the expected percentage of red balls remaining in the box is  $\frac{r}{N}$ . If no green balls have been drawn, what is a reasonable estimate for the number of green balls remaining in the box?

If the fraction of green balls in the box is  $g$ , the probability of not having drawn a green ball is  $(1 - g)^N$ . The 95% confidence bounds on this occurring is:  $(1 - g)^N \leq 0.05$ .

$$N \log(1 - g) \leq \log(0.05) \approx -Ng \leq -2.9957 \approx g \leq \frac{3}{N}$$

The non-appearance of any green balls suggests that  $g$  is very small, and so  $\log(1 - g) \approx -g$ .

## 9.1.2 Measurement scales

Mathematically, measurement values can be characterised as discrete or continuous, along with the properties of the scale used. Possible scales include the following:

- Discrete

– *nominal scale*: each measurement value has an arbitrary number or name. Because the choice of number/name is arbitrary, no ordering relationship exists between different numbers/names. A nominal scale is not a scale in the usual sense of the word.

Examples: the numbers on the back of footballers' shirt or the various sales regions in which a product is sold.

- *ordinal scale*: each measurement value is a number or name of an item and an ordering relationship exists between the numbers/names. The distance between distinct values need not be the same. When names are assigned to entities, there may be cultural differences in the selection process. Figure 9.2 shows how words are assigned to tracts of trees having occupying various surface areas.

Example: Classifying faults by their severity, e.g., minor, moderate, serious.

If a minor fault is considered less important than a moderate fault, and a moderate fault is less important than a serious fault, we can deduce that a minor fault is less important than a serious fault.

Example: The addresses of members of a C structure type is increasing, for successive members, but the difference between member addresses is not fixed because different members can have different types.

- Continuous

- *interval scale*: each measurement is a number, a relative ordering exists and a fixed length interval of the scale denotes the same amount of quantity being measured.

A data point of zero does not indicate the absence of what is being measured.

Example: the start date of some event is an interval scale. If the start date of events A, B and C are known, and the difference in start date between events A and B is the same as between events C and D, it is possible to calculate the start date of event D.

Addition and subtraction can be applied to values on an interval scale but not multiplication or division (e.g., it makes no sense to say that the start date of event A is twice that of event C).

- *ratio scale*: each measurement assigns a number to an item and this numeric scale preserves: the ordering of items, the size of the interval between items and the ratios between items. It differs from the interval scale in that a measurement of zero denotes the lack of the attribute being measured.

The time difference between two events is a ratio scale.

The kinds of statistical analysis that can be legitimately performed on the values in a sample will depend on the kind of measurement scale used.

## 9.2 Probability distributions

Probability distributions are mathematical descriptions of the properties of values calculated by following a pattern of behavior (i.e., an algorithm). For instance, the flipping a coin pattern of behavior generates one of two results, a fixed probability of either result, with each result being independent of the previous one and a count of the number of heads and tails has a binomial probability distribution.

If a sample of values can be fitted to a known probability distribution, then information about the pattern of behavior that generated them can be inferred from what is known about processes known to generate values having that particular distribution. For instance, given a list of pairs of numbers, if the ratio formed from each pair (i.e.,  $\frac{a}{a+b}$ ) can be fitted to a binomial distribution, there is strong evidence that the pairs are counts of a process producing one of two possible values (e.g., heads/tails, yes/no etc) and the probability of producing each value can be calculated from the fitted distribution.

While many probability distributions have been created,<sup>422</sup> only a handful of them are regularly used by analysts; R packages tend to support commonly occurring distributions, with a few packages supporting a wide range of distributions.<sup>422</sup>

Fitting a distribution to a sample is a step towards understanding the processes that generated the measurements, not an end in itself.

Failure to fit a known distribution may mean that more than one distribution is involved, e.g., two different coins are being used and both are biased in some way. Given enough data it is sometimes possible to obtain a reasonable fit that involves two or more distributions.

If there is reason for believing the processes being measured are driven by a known behavior, the quality of fit of the predicted probability distribution to the measured values can be compared; perhaps also against the quality of fit to other distributions.

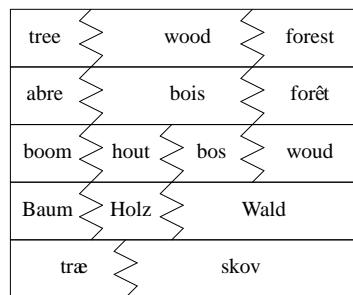


Figure 9.2: The relationship between words for tracts of trees in various languages. The interpretation given to words (boundary indicated by the zigzags) in one language may overlap that given in other languages. Adapted from DiMarco et al.<sup>405</sup>

If there is no expectation of a particular behavior, then finding an acceptable fit of some probability distribution to the measurement values is a starting point for understanding the processes that are driving the measurements observed.

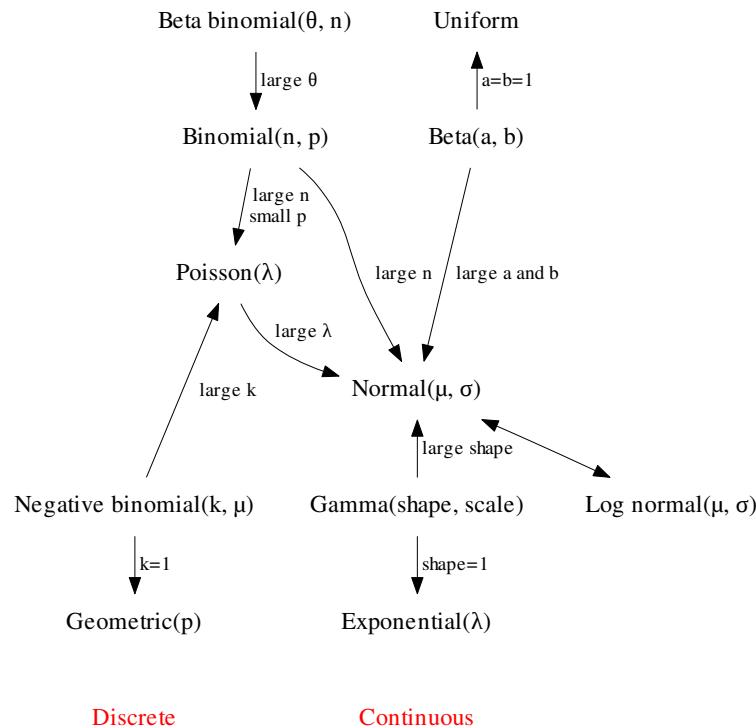


Figure 9.3: Relationships between commonly used discrete and continuous probability distributions.

Every family of probability distributions is completely characterised by a small set of numbers (often one or two) and a formula that the numbers parameterise. For instance, everything about a Normal probability distribution can be calculated by plugging values for the mean,  $\mu$ , and standard deviation,  $\sigma$ , into the formula:  $P(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$  (this formula is often abbreviated as  $N(\mu, \sigma)$ ). Fitting data to a Normal distribution involves finding appropriate values for  $\mu$  and  $\sigma$ .

In practice, a few probability distributions are encountered much more often than others. One of these common cases will often fit reasonably well to a wide spectrum of commonly encountered samples, and unless there are theoretical reasons for expecting one of the less commonly encountered distributions, there is nothing to be gained by searching through all known distributions to find the one that best fits a sample.

Some of the characteristics of sample values, that may or may not correspond to a known probability distributions include:

- mean value (sometimes called the *arithmetic mean*, *central tendency* or *location value*, the last two terms may also be used to refer to the median): many distributions have a finite mean (examples that don't include power laws with an exponent greater than or equal to  $-1$  and the Cauchy distribution),
- scale parameter, *variance* (*standard deviation* is the square-root of variance): how spread out the distribution is; a few distributions do not have a finite variance, e.g., power laws with an exponent between zero and  $-2$ ,
- the extent to which the distribution is symmetrical/asymmetrical about its mean, the *skew* of a distribution is a measure of how asymmetrical it is; a symmetric distribution has a skew of zero, while a positive skew has a tail pointing towards larger positive values and a negative skew has a tail pointing towards negative values,
- where most of a distribution's density resides, e.g., around the mean or in the tails. The *kurtosis* of a distribution is a measure of how spiky the distribution is; possibilities include tall and slim (known as *leptokurtic*; slender-curved), short and flat (known as *platykurtic*) or medium-curved (known as *mesokurtic*; the Normal distribution has a Kurtosis of three),

- number of distinct peaks, known as the *modality* of a distribution; a distribution with one distinct peak is said to be unimodal, two distinct peaks bimodal (such as measurements from two different distributions, e.g., height of men/women).

The `moments` package contains functions for calculating skewness, kurtosis and moment related attributes of a numeric vector.

Probability distributions can be divided into discrete and continuous distributions, with discrete distributions only being defined at specific points (usually integer values). In R, functions that involve discrete distributions usually require integer values while functions involving continuous distributions take floating-point values.

There are various ways of representing a probability distribution and the following are often encountered:

- density function: for discrete distributions (see Figure 9.4) this can be viewed as the probability that  $x$  will have a given *value*,  $P(x = \text{value})$ ; for continuous distributions (see Figure 9.6) the probability of any particular value occurring is zero, however there is a finite probability of a measurement returning a value within a specified interval,
- cumulative density function: the probability that  $x$  will be less than or equal to a given *value*,  $P(x < \text{value})$ , see Figure 9.5,
- equation: an equation for the probability distribution. For the majority of people (including your author), this is little more than eye candy, e.g., the equation,  $\frac{\lambda^k e^{-\lambda}}{k!}$ , is very difficult to visualize and is only of use to developers wanting to implement the Poisson distribution.

**Discrete distributions:** commonly encountered discrete distributions include the following (see Figure 9.4):

- *Binomial distribution*: for a random variable  $X$ ,
  1. the process involves a sequence of independent trials,
  2. each trial produces two possible outcomes, e.g., heads/tails,
  3. the probability of either outcome ( $p$ , say, for heads) does not change,
  - $X$  counts the number of success (where success might be defined as a head occurring) in  $n$  fixed trials.

The Binomial distribution is completely described by two parameters:  $B(n, p)$ , This process is sometimes described using the analogy of, drawing  $n$  objects from a pool containing a finite number of two kinds of object, where the object is placed back in the pool after it has been drawn (this kind of draw is said to be: *with replacement*). The Hypergeometric distribution is the result, if objects are not returned to the pool once they are drawn (this kind of draw is said to be: *without replacement*).

A distribution supporting more than two discrete values is known as a *Multinomial distribution* (again with a fixed probability of each value occurring). The `XNomial` package provides support for multinomial distributions,

- *Negative Binomial distribution*: this has the same three requirements as the Binomial distribution, but differs in what is counted,
  - $X$  counts the number of trials up to and including the  $k^{th}$  success (where success might be defined as a head occurring after a continuous sequence of tails).

A process that produces values having a Negative Binomial distribution is randomly drawing from a mixture of Poisson distributions, where the mean of the mixture of Poisson distributions has a Gamma distribution,

This distribution is a generalised version of the Geometric distribution (which is based on the probability of observing the first success on the  $n^{th}$  trial).

- *Poisson distribution*: for a random variable  $X$ ,

1. the process involves independent events,
2. only one event can occur at any time,

- $X$  counts the number of events that occur within a specified time.

The Poisson distribution is completely described by one parameter ( $\lambda$ , the distribution mean):  $P(\lambda)$ ,

The sum of two independent Poisson distributions  $P(\lambda_1)$  and  $P(\lambda_2)$  is the Poisson distribution:  $P(\lambda_1 + \lambda_2)$ .

The Binomial and Poisson distributions are related in that as  $n \rightarrow \infty$  and  $p \rightarrow 0$ , then  $B(n, p) \rightarrow P(np)$ , i.e., The Poisson distribution is a limit case of a Binomial distribution having a very low probability of success over a long period.

**Continuous distributions: commonly encountered continuous distributions** include the following (see Figure 9.6):

In all but one case, the generating process clusters the values around a single peak.

- *Uniform distribution*: all values between the lower and upper bounds of the interval have an equal probability of occurring, i.e., no value is more likely to occur than any other. For discrete values between 1 and  $n$  the probability of any value occurring is  $\frac{1}{n}$ .

One process that generates a uniform distribution is a random number generator, such as calling &R's `runif` function.

- *Normal distribution*: can be generated by adding together contributions from many independent processes; a consequence of the Central limit theorem. This distribution crops up with great regularity, it has a mathematical form that is easier to manipulate analytically, than many other distributions, resulting in it being widely used before computers reduced the need for analytic solutions to equations. This distribution is described by its mean and variance.

While the Normal distribution is the result of adding contributions from many independent processes, it is not true to say that adding contributions from many different kinds of processes will result in this distribution (similarly, for multiplicative contributions and a lognormal distribution). For instance, given the right conditions, adding values drawn from many different Poisson distributions can result in a Negative Binomial distribution, a Geometric distribution and many other distributions,<sup>825</sup>

- *Lognormal distribution*: the logarithm of a Normal distribution, which can be thought of as being generated by multiplying together the sum of contributions from many independent processes;<sup>1095</sup> samples drawn from a Lognormal distribution can produce a straight line, over some of their range, when plotted using log-log axis,

- *Exponential distribution*: generated by a memoryless process, e.g., the waiting time for an event to occur is independent of the amount of time that has passed since the last event. This is the continuous form of the Geometric distribution, and like it, is described by a single parameter.

Over some of its range the exponential distribution is visually similar to a Power law, which has led researchers to incorrectly claim that their sample fits a power law (a fashionable distribution to have one's sample following).

The sum of a reasonably large number of independent exponential distributions has an Erlang distribution, e.g., the interval between incoming calls to a telephone exchange, where the interval between calls from any individual have an exponential distribution,

- *Beta distribution*: applies to processes where the explanatory variable is restricted to a finite interval, e.g., the interval zero to one. This distribution is defined by two, non-negative, shape parameters.

- *Gamma distribution* ( $\Gamma$  is the Greek uppercase Gamma, the symbol often used to denote the Gamma function, is the lowercase version,  $\gamma$ ): used to describe waiting times, e.g., `Gamma(shape=3, scale=2)` is the distribution of the expected waiting time (in some units) for three events to occur, given that the average waiting time is 2 time units (yes, the Gamma function differs from most other distribution names in the base system by starting with an uppercase letter).

When `shape=1`, the Gamma distribution reduces to the Exponential distribution.

The Gamma distribution is the continuous equivalent of the Negative binomial distribution.

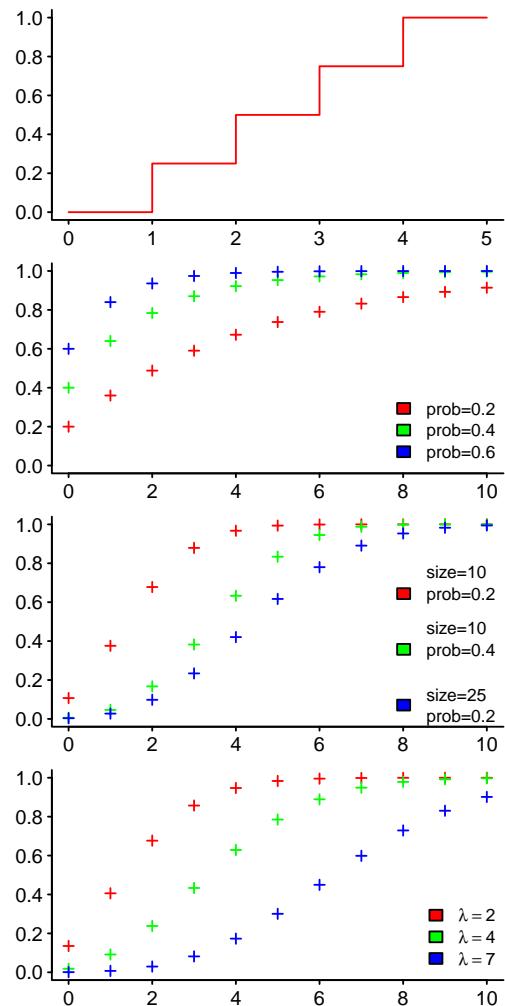


Figure 9.5: Cumulative density plots of the discrete probability distributions in Figure 9.4. [code](#)

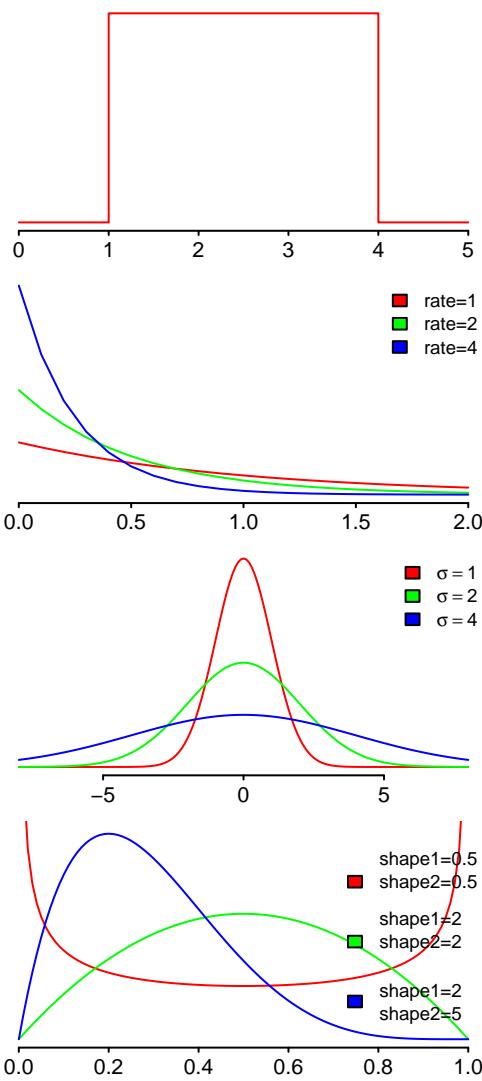


Figure 9.6: Commonly encountered continuous probability distributions (upper to lower: Uniform, Exponential, Normal, beta). [code](#)

- *Chi-squared distribution* (sometimes written using  $\chi$ , the Greek lowercase letter of that name): is more often encountered in the mathematical analysis of statistics, than as a distribution of a sample. A random variable has a chi-squared distribution, with  $d$  degrees of freedom, if it is produced by a process which generates the values:  $Z_1^2 + Z_2^2 + \dots + Z_d^2$ , where  $Z_i$  are independent random variables having a Normal distribution.

The chi-squared distribution is a special case of the Gamma distribution.

- *Weibull distribution*: this distribution drops out as the solution to various problems in hardware reliability, e.g., time to failure, and is often used as the hazard function in survival analysis. The Exponential and Rayleigh distributions are special cases of the Weibull distribution,

- *Cauchy distribution*: this distribution is more famous for its unusual characteristics, e.g., having an undefined mean and variance (because of its very fat tail), than through its uses. The density function for the average of two random variables each having a Cauchy density is a random variable with a Cauchy density; this self mapping is unique to the Cauchy distribution. One consequence is that, if the error in a measurement has a Cauchy density, then the average of many measurements will not be more accurate than the individual measurements.

### 9.2.1 Comparing probability distributions for equality

The mathematics of comparing sample distributions is framed in terms of comparing the measurable characteristics of probability distributions, but the practical question of interest is whether two or more samples are drawn from the same population. The measurable characteristics include mean, standard deviation, skew and kurtosis (a probability distribution is defined via a formula having parameters that get plugged into it to create a specific instance; a formula is not measurable).

If we are interested in the distribution as a whole, not individual characteristics, the various comparison methods are based on some measure of difference between the *shape* of the sample distributions.

As always, visualization is a useful first step in judging whether two samples might be drawn from the same distribution. However, be warned, small datasets can produce visualizations showing little resemblance to the distribution from which they were drawn; as can be seen from Figure 9.7, where all the samples are drawn from the same Normal distribution.

A study by Veytsman and Akhmadeeva<sup>1606</sup> measured subject reading rate, in words per minute, for text printed using a Serif or Sans Serif font. Words per minute is a discrete distribution and subject performance is likely cluster around similar values, i.e., there will be duplicates. Figure 9.8 shows a density plot of the normalised data.

The following tests are based on comparing the edf (empirical distribution function) of the samples.

- The Anderson-Darling test is based on the largest difference between the edf of the two distributions, it uses weights to ensure that the tails of the distribution have as much influence as other parts of the distribution; it is possible to use this test to compare more than two distributions. While the Kolmogorov-Smirnov test is often encountered, it has been found to be less sensitive than the Anderson-Darling test<sup>1503</sup> because it primarily detects differences in the main body of the distribution, rather than over the complete range of values.

The `ad.test` function in the `kSamples` package implements the Anderson-Darling test for two or more samples.

The `ks.test` function, part of the base system, implements the Kolmogorov-Smirnov test; other implementation include the `ks.test` function in the `dgof` package whose interface is the same but includes support for discrete distributions.

Samples drawn from a continuous distribution are very unlikely to contain identical values and many implementations warn if a sample contains duplicate values.

- The Cramér-von Mises test is based on summing (the square of) differences between edfs, rather than using a single maximum value, and can be more powerful against a large class of alternative hypothesis.<sup>58</sup>

The `cvm.test` function in the `dgof` package implements the Cramér-von Mises test.

The choice of statistical test depends on whether differences over the range of values in the samples are of interest, whether tail values are uninteresting (perhaps because there are few measurements in the tail and so what is there is noisy) or the amount of difference between samples is the primary differentiator.

Comparison of samples drawn from discrete distributions is provided by the WRS package (on Github), which implements a version of the Kolmogorov-Smirnov test (the ks function) that supports discrete data and also the bmpmul function which uses the Brunner-Munzel test (also see the ks.test function in the dgof package).

The following code performs various tests that check whether the two sample are likely to have been drawn from the same population (see [group-compare/tb104veytsman-dist.R](#)):

```
library("dgof")
library("kSamples")
library("WRS")

# From WRS
ks(serif$Standard_WPM, sansserif$Standard_WPM)
# In fact unscaled measurements give the same result, i.e., not different
ks(serif$WordsPerMinute, sansserif$WordsPerMinute)

dgof::ks.test(serif$Standard_WPM, ecdf(sansserif$Standard_WPM))

# From base system
ks.test(serif$Standard_WPM, sansserif$Standard_WPM)

# Only applicable to continuous distributions
ad.test(serif$Standard_WPM, sansserif$Standard_WPM)
```

The hypothesis that the samples plotted in Figure 9.8 are drawn from populations having different distributions is rejected.

Note that many measurement points may be needed to reliably detect a difference in distributions, when one exists. For instance, when one sample is drawn from an Exponential distribution and the other from a Normal distribution, two samples of 150 points are needed to obtain a 95% confidence level, using ad.test, that the samples are drawn from different distributions (550 points are needed when the samples are drawn from Normal and Uniform distributions); see [group-compare/ad-check.R](#).

For some analysts, testing whether a sample is drawn from a Normal distribution is a common activity (techniques that are practical to perform manually often require that samples be drawn from this distribution<sup>iii</sup>).

The result of testing whether a small sample is drawn from a Normal distribution has a high degree of uncertainty. The points in Figure 9.9 was obtained by testing samples, all drawn from the same distribution (e.g., via a call to rexp), using the shapiro.test function (replicated 1,000 times for each sample size). The y-axis shows the probability of the Shapiro-Wilk test detecting that the sample values are not drawn from a Normal distribution ( $p\text{-value} < 0.05$ ; when the values have been drawn from another distribution); for the case when the values are drawn from a Normal distribution (e.g., a call to rnorm) the y-axis gives the probability of this fact not being detected.

There is no guarantee that the values in a sample have a distribution that even closely resembles any of the known probability distributions.

A study by Berger, She, Czarnecki and Wąsowski<sup>146</sup> investigated the use of feature macros used in the configuration of software product lines. Figure 9.10 shows the number of conditionally compiled sections of source code that were dependent on a given number of feature macros.

A Cullen and Frey graph shows that the characteristics of neither sample are close to matching any of the common discrete distributions. A Kolmogorov-Smirnov test considers them to be sufficiently different, that they are likely to have been drawn from different distributions (see [group-compare/cond-compile/2010-berger.R](#)).

Samples may appear to have a similar shape, but have different mean values. Technically, samples with different mean values (or standard deviations) are considered to be drawn from

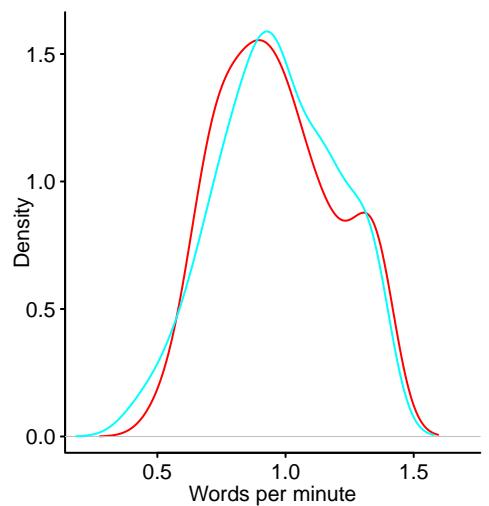


Figure 9.8: Reading rate for text printed using a serif (blue) and sans-serif (red) font, data has been normalised and displayed as a density. Data from Veytsman et al.<sup>160</sup> code

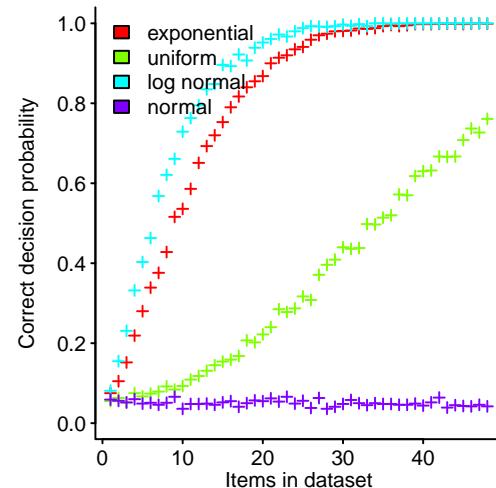


Figure 9.9: Probability, with  $p\text{-value} < 0.05$ , that shapiro.test correctly reports that samples drawn from various distributions are not drawn from a Normal distribution, and probability of an incorrect report when the sample is drawn from a Normal distribution; 1,000 replications for each sample size. [code](#)

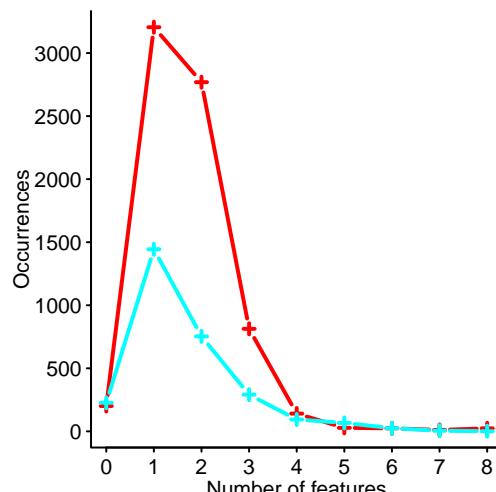


Figure 9.10: Number of conditionally compiled code sequences dependent on a given number of feature macros (red overwritten by blue: Linux, blue: FreeBSD). Data from Berger et al.<sup>146</sup> [code](#)

<sup>iii</sup> Readers of this book learn techniques that don't have this precondition

different distributions. There may be theoretical reasons for believing that samples have been generated by the same processes and normalizing mean values (or even variance) enables the shape of the sample distributions to be compared.

A study by Zhu, Whitehead, Sadowski and Song<sup>[1712](#)</sup> counted the number of various kinds of statements in a corpus of C, C++ and Java programs (approximately 100 programs, around 10 million lines, for each language). Figure 9.11 shows the distribution of occurrence (expressed as a density on the y-axis) of various statements (expressed as a percentage on the x-axis), over the programs measured; a different color for each language, figure out which is which, before looking at the code.

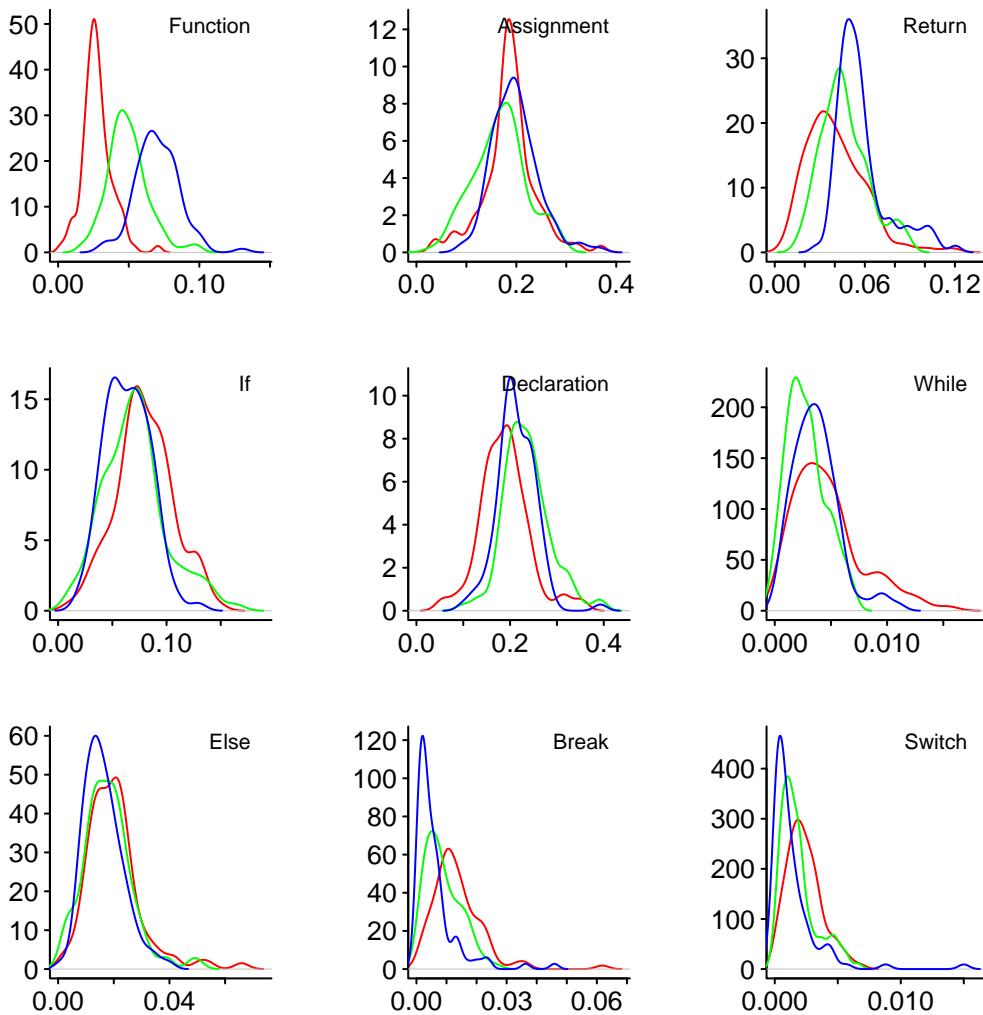


Figure 9.11: Percentage occurrence of statements for each of 100 or so C, C++ and Java programs, plotted as a density on the y-axis. Data from Zhu et al.<sup>[1712](#)</sup> code

Differences in the probability of various kinds of statements being used, over a sample of programs written in various languages, is evidence that language has an impact on what code gets written (either because particular kinds of applications are written using a given language, particular algorithm selection is influenced by language, or the impact of differences in language semantics).

Might two or more of the languages measured be said to have the same distribution of if-statement and/or assignment-statement usage? The interactions between different statements makes the analysis non-trivial.

The takeaway from this section is that for small sample sizes, distribution comparison produces unreliable answers, and for large samples comparison may be complicated.

Comparison of particular characteristics of sample distributions, e.g., sample means, is covered in Chapter 13.

### 9.3 Fitting a probability distribution to a sample

Given a sample of values, which of the known, supported by R<sup>[422](#)</sup> probability distributions is the best fit?

There is no universal best-test statistic, for goodness-of-fit of a sample to a probability distribution. The performance of the available tests depends on the (unknown) distribution from which the sample was drawn.<sup>1496</sup>

The Normal distribution is often the default answer given, when people are asked about the distribution of a sample. There are several reasons for this, including: historically many of the techniques designed to be performed by a human calculator were derived from theory that assumed normally distributed data (which often appeared to work reasonably well, when the data only approximated a Normal distribution), along with a misunderstanding of what the Central Limit theorem is about, driving a belief that a complex process provides the mixing needed to produce a Normal distribution.

As always, knowledge of the processes driving the production of the measured values can be very useful. For instance, measurements of arrival times that are driven by a Poisson process will result in inter-arrival times that are exponentially distributed, values created via the multiplicative effect of many contributions may have a Lognormal distribution and a preferential attachment process often results in links or what they link-to following a power law.

If there is no theoretical justification for a particular distribution, limiting the selection process to those distributions having some degree of name recognition is likely to make the one chosen an easier sell to readers. For instance, the Delaporte distribution<sup>iv</sup> might happen to fit a particular sample slightly better than the Negative Binomial distribution, but its lack of name recognition means that extra effort will have to be invested, justifying its use.

A study by van der Meulen<sup>1591</sup> posted the  $3n + 1$  problem on a programming competition website: 95,497 solutions were submitted and van der Meulen kindly sent me a copy of these solutions (11,674 solutions were written in Pascal, the rest in C). The  $3n + 1$  problem is: write a program that takes a list of integers and outputs the *length* of each value, where length is the number of iterations of the following algorithm:

```
for input integer ++pass:[n]++
  while (n != 1)
    n = (is_even(n) ? n/2 : n*3+1);
```

Which distribution is a good approximation, to the number of lines of code contained in the programs submitted as answers to this problem?

The first step of visualizing the sample provides basic information about the shape of the distribution, e.g., decreasing/increasing, single/multiple peak, symmetric/skewed or appearing to be nothing but random noise (see Figure 9.13).

A method of narrowing down the list of possible distributions, is to plot a Cullen and Frey graph. The `descdist` function, in the `fitdistrplus` package, plots this graph and returns some descriptive distribution characteristics of the values (mean, median, sd, skewness and kurtosis). Skew and kurtosis are not reliable estimators and `descdist` includes an option to create and test bootstrap samples.

The blue circle and yellow points in Figure 9.12 denote the sample and various bootstrapped results for the  $3n + 1$  program lengths, assuming a continuous distribution (the average number of lines is large enough that the difference between discrete/continuous is likely to be small). The sample does not overlay any of the grey lines/areas on the plot that denote commonly occurring distributions. The code is:

```
library(fitdistrplus)

# Default is to check continuous distributions
# dummy=descdist(li, discrete=TRUE, boot=500)
dummy=descdist(li, boot=500)
```

The `fitdist` function<sup>v</sup> in the `fitdistrplus` package can be used to fit a distribution to the data, i.e., find values of the specified distribution's parameters, such as mean and variance, that minimise some measure of goodness-of-fit (the AIC of the fit is returned). The `gamlss` package supports a wider range of distributions (see the help information for the `gamlss.family` function) that `fitdist` can use to fit data.

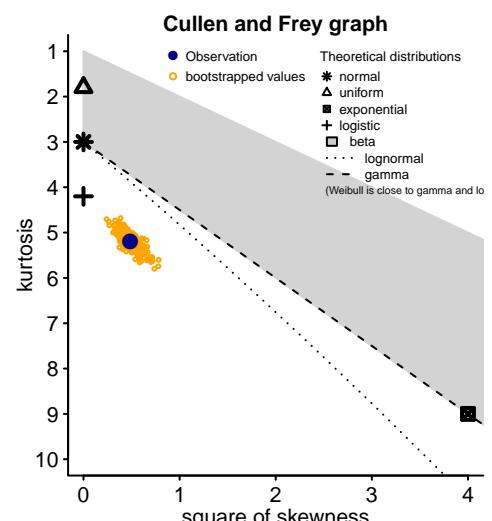


Figure 9.12: A Cullen and Frey graph for the  $3n + 1$  program length data. Data kindly provided by van der Meulen.<sup>1591</sup> code

<sup>iv</sup> A compound distribution derived from a Poisson distribution whose mean has a shifted Gamma distribution.

<sup>v</sup> The MASS package contains the `fitdistr` function and the `gamlss` package contains the `fitDist` function, both of which fit distributions to data.

Figure 9.13 shows fits for the Normal, Poisson, Lognormal and Negative binomial distributions.

```
library(fitdistrplus)

tp=fitdist(li, distr="pois"); tnb=fitdist(li, distr="nbinom")
tn=fitdist(li, distr="norm"); tln=fitdist(li, distr="lnorm")

# gofstat is a way of getting all the values used for plotting
theo_vals=gofstat(list(tn, tp, tln, tnb), chisqbreaks=1:120,
                  fitnames=c("Poisson", "Negative binomial",
                            "Normal", "Lognormal"))

plot_distrib=function(dist_num)
{
  lines(theo_vals$chisqbreaks, head(theo_vals$chisqtable[, 1+dist_num], -1),
         col=pal_col[dist_num])
}

plot(theo_vals$chisqbreaks, head(theo_vals$chisqtable[, 1], -1), type="h",
      xlab="Program length", ylab="Number of programs\n")
plot_distrib(1); plot_distrib(2)
plot_distrib(3); plot_distrib(4)
```

The large spike at 50 lines might be caused by solutions all doing the same thing, but with different statement orderings, e.g., multiple submissions derived from a common solution.

Based on minimizing AIC, the Normal distribution is the best fit, with the Negative binomial distribution a close second. Should either distribution be chosen as the best fitting, or is it worthwhile attempting to fit other distributions? The answer depends on what the fitted distribution will be used for, e.g., making predictions or building models. Jumping to any conclusions based on one data-point (i.e., set of length measurements for one problem) is always problematic.

### 9.3.1 Zero-truncated and zero-inflated distributions

Some distributions only make use of non-negative values, they start at zero, e.g., the Poisson distribution. While zero is a common lower bound for measurement values, other lower bounds occur, e.g., the number of minutes to complete a task (the zero time tasks, that are never started, are not measured).

It is possible to adjust the equations that describe zero-based distributions, to have a non-zero lower bound. Rebasing a distribution to start at one (rather than zero) is the common case and after such an adjustment the distribution is said to be *zero-truncated*, e.g., *zero-truncated Poisson distribution*.

The `gamlss.tr` package contains functions that support the creation of zero-truncated (or truncation to the right or left of any value) distribution functions. The following code creates a set of functions relating to the zero-truncated type II Negative binomial distribution; the name of the created function is `NBIItr` and like other distribution functions in R, the associated density, distribution, quantile and random functions are obtained by prefixing the letters `d`, `p`, `q` and `r`, respectively, to `NBIItr`:

```
library(gamlss)
library(gamlss.tr)

gen.trun(par=0, family=NBII) # Bring various functions into existence
```

The 7Digital data<sup>1</sup> (discussed in more detail in Section 5.4.5) contains information on 3,238 features implemented between April 2009 and July 2012; the information consists of three dates (Prioritised/Start Development/Done), from which a non-zero duration can be calculated.

The Cullen and Frey graph suggests a negative binomial distribution might be a good fit.

The functions returned by `gen.trun` do not have a form that can be used in calls to the `fitdist` function. The `gamlss` function in the `gamlss.tr` package has a special form for

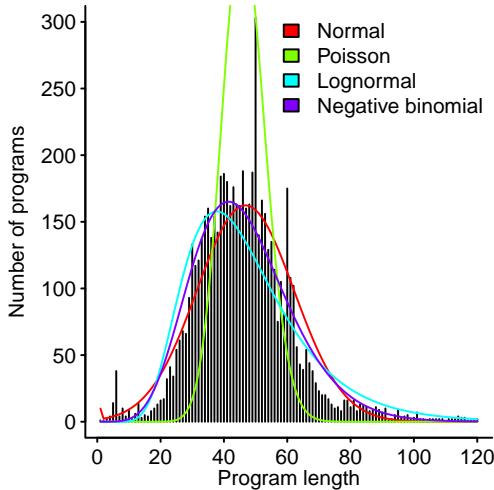


Figure 9.13: Number of 3n+1 programs containing a given number of lines and four distributions fitted to this data. Data kindly provided by van der Meulen.<sup>1591</sup> [code](#)

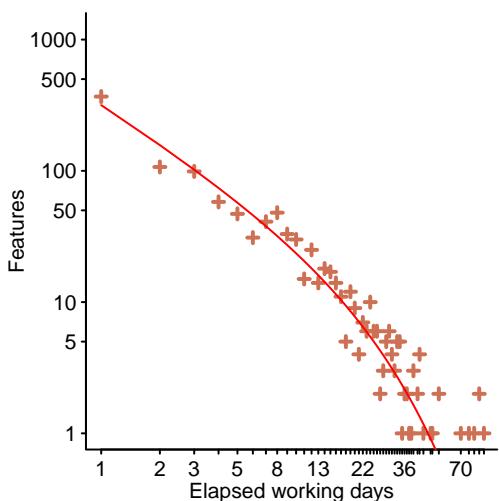


Figure 9.14: A zero-truncated Negative Binomial distribution fitted to the number of features whose implementation took a given number of elapsed workdays; first 650 days used. Data kindly provided by 7digital.<sup>1</sup> [code](#)

handling these created functions, as shown in the following code (where `day_list` contains the list of values and `NBIItr` was created by an earlier call to `gen.trun`). The following code was used to produce Figure 9.14:

```
library(gamlss)
library(gamlss.tr)

g.NBIItr=gamlss(day.list ~ 1, family=NBIItr)

NBII.mu=exp(coef(g.NBIItr, "mu"))      # get mean coefficient
NBII.sigma=exp(coef(g.NBIItr, "sigma")) # standard deviation

plot(table(day.list), log="xy", type="p", col=point_col,
     xlab="Elapsed working days", ylab="Features\n")

lines(dNBIItr(1:93, mu=NBII.mu, sigma=NBII.sigma)*length(day.list), col="red")
```

One process generating values having a Negative binomial distribution is based on a mixture of Poisson distributions, whose means have a Gamma distribution. It is possible to generate other distributions by combining a mixture of Poisson distributions, are any of these a better fit of the data? The Delaporte distribution sometimes fits slightly better and sometimes slightly worse (see Section 6.4.2 for details); the difference is not large enough to warrant switching from a relatively well-known distribution, to one that is rarely covered in text books or supported in software; if data from other projects is best fitted by a Delaporte distribution, then it may be worthwhile spending time analysing how this distribution might be a better model of project scheduling.

If the processes generating these values can be modeled by a mixture of Poisson distributions, there is unlikely to be a single subprocess responsible for a large percentage of the quantity measured, many subprocesses are involved.

Sometimes count data contain many more zero values than are expected, from the distribution that the generating process is believed to follow. Two kinds of behavior that can cause an excess of zeroes to appear in the measurements are:

- a process that generates zeroes and a process that generates non-negative values; this situation can be modeled by what is known as *zero-inflated model*. The `gamlss` package supports zero-inflated distributions,
- the measurements involve two processes, one where the values are zero or non-zero, and the other where values are always non-zero (i.e., zero-truncated); this situation can be modeled by what is known as a *hurdle model* (the hurdle that has to be got over is moving from zero to non-zero). The `gamlss` package supports what it calls *zero altered* (or *zero adjusted*) distributions; the `pscl` package uses the term *hurdle*.

A search for software engineering measurements containing an excess of zeroes located a few that appeared to contain such an excess, but none could be fitted by the models discussed above (see [probability/bolz\\_data\\_struct\\_racket.R](#) for an example).

### 9.3.2 Mixtures of distributions

Sometimes sample measurements are generated by two or more distinct processes, resulting in values that appear to be drawn from two or more distinct distributions, e.g., a plot shows multiple peaks. A model built using a mixture, or weighted sum, of distributions is known as a *finite mixture model* or just a *mixture model*; a continuous mixture of distributions is known as a *compounded distribution* (the Negative Binomial distribution is a compounded distribution).

The `mixtools` and `rebmix` packages contain functions for fitting samples drawn from two or more of the same kind of distribution family, e.g., multiple Normal distributions. The two packages differ in the structure of their API, e.g., one having many functions and the other having one main function taking many arguments (neither would win a prize for user interface design).

A study by Hunold, Carpen-Amarie and Träf<sup>740</sup> investigated the impact of external factors on the performance of an MPI micro-benchmark. Figure 9.15 shows the runtime variation

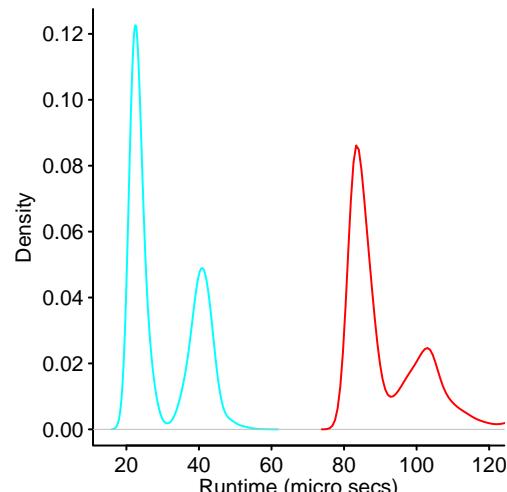


Figure 9.15: Density plot of MPI micro-benchmark runtime performance for calls to `MPI_Allreduce` with 1,000 Bytes (left curve) and to `MPI_Scan` with 10,000 Bytes (right curve). Data kindly supplied by Hunold.<sup>740</sup> [code](#)

of two different MPI calls, with each having two distinct peaks. The two peaks in the left curve appear to be symmetrical and perhaps a mixture of two Normal distributions is a good fit. Figure 9.16 shows the two distributions fitted by a call to the `normalmixEM` function (in the `mixtools` package), along with a histogram (all produced by the same call to the `plot` function provided by the package).

```
library("mixtools")
scan_dist=normalmixEM(fig1_Allreduce$time)
plot(scan_dist, whichplots=2, main2="", col2=pal_col,
      xlab2="Time (micro secs)", ylab2="Density\n")
```

A call to `summary` returns the parameters of the fitted model; the first row (prefixed by `lam bda`) is the fraction contributed by each distribution, followed by the mean, standard deviation and log likelihood (rather than AIC): `code`

```
number of iterations= 15
summary of normalmixEM object:
      comp 1      comp 2
lambda 0.611002  0.388998
mu     23.011364 40.703293
sigma   1.720527  3.378665
loglik at estimate: -28873.39
```

A plot of a sample drawn from a mixture of distributions does not always have visually distinct peaks; if  $f_1$  and  $f_2$  are normal densities with means  $\mu_1$  and  $\mu_2$ , respectively, and both have the same variance  $\sigma^2$ , then the mixture density  $f = 0.5f_1 + 0.5f_2$  will have a single peak if, and only if:  $abs(\mu_2 - \mu_1) \leq 2\sigma$ .

A study by Kaltenbrunner, Gómez, Moghnieh, Meza, Blat and López<sup>813</sup> analysed the pattern of user activity of the Slashdot technical community news site. The black curve in Figure 9.17 shows the density of the number of accesses to one article in each minute after first publication (a total of 1,567 accesses).

A possible explanation for the multiple upticks in number of accesses, is the article being linked to from other web sites, driving a fresh batch of readers to Slashdot. Which mixture of distributions might best fit the access times of this Slashdot article? The Poisson distribution is often used to model arrival times and is the obvious first choice, but in this particular case turns out not to provide the best fit.

Figure 9.17 shows several Normal distributions fitted to data, on a log scale, using functions from the `rebmix` and `mixtools` packages. The algorithms used by packages do not guarantee to find the globally optimal solution and differences in the mix of distributions selected can occur because of differences during the search process.

```
library("rebmix")
slash_mod=REBMIX(Dataset=list(data.frame(users=log(slash$users))),
                  Preprocessing="histogram", cmax=5,
                  Variables="continuous", pdf="normal", K=7:45)

plot_REBMIX_dist=function(dist_num)
{
  y_vals=dnorm(x_vals, mean=as.numeric(slash_mod$Theta[[1]][2, dist_num]),
                sd=as.numeric(slash_mod$Theta[[1]][3, dist_num]))
  lines(x_vals, slash_mod$w[[1]][1, dist_num]*y_vals, col=pal_col[dist_num])
}

plot(work_den, main="", xlim=c(0, 10), ylim=c(0, 0.36),
      xlab="", ylab="Access density\n")
plot_REBMIX_dist(1); plot_REBMIX_dist(2)
plot_REBMIX_dist(3); plot_REBMIX_dist(4)
```

Fitting a Normal distribution to log scaled data means that the sample actually has a Lognormal distribution. Is the Lognormal distribution a good representation of the processes driving readers to access Slashdot articles? As always in model building the answer depends on what

the model is to be used for. If the purpose is to make predictions, the accuracy of prediction is of more interest than any underlying assumptions; if the purpose is to understand what is going on, then a theory containing processes generating Lognormal distributed behavior is needed.

It can take a lot of analysis, over many years, to settle on the distribution, or combinations of distributions, that best describes the measured properties of a system. The study of file-system characteristics<sup>15</sup> is an example of how researchers' ideas and models changed over time,<sup>412, 756, 1096</sup> becoming more sophisticated as more data became available, from various platforms,<sup>363</sup> and more analysis was performed.

### 9.3.3 Heavy/Fat tails

*Heavy tailed* is the term used to describe distributions where the majority of values occur a long way from the mean value (*fat tails* and *long tail* are also used).<sup>vi</sup> When the 80/20 rule applies the distribution is heavy tailed and the frequency with which this rule is encountered suggests that such data is not rare. The `poweRlaw` package supports operations involving a variety of heavy tailed distributions, including power laws.

Averaging the performance of multiple subjects can produce values that are well fitted by a power law, while individual subject performance is well fitted by any of a variety of other distributions.<sup>1120</sup>

The Pareto distribution is the mathematical name of a particular instance of a heavy tailed distribution (sometimes going by the name *power law* in popular culture); Zipf's law is a particular instance of this distribution.

The mean value of a heavy tailed distribution may not exist (because it is infinite). Any finite dataset has a finite mean, and if a sample is drawn from a heavy tailed distribution, its mean value will jump around erratically.

It is more difficult to narrow down a distribution which best fits a sample drawn from a heavy tailed distribution (because several fit equally well), compared to one without a heavy tail; a sample may contain many values, but their density may be low because values are spread out over a long tail (rather than in a high density cluster around a central location).

Figure 9.18 is from a survey<sup>756</sup> of file sizes and shows that a small percentage of files account for most of the disk space occupied (the vertical line meets the bytes line where 89.9% of disk space has yet to be consumed and the files line where 12.5% of files still remain to be accounted for). Another way of describing the situation is to say that there is a mass/count disparity, i.e., a few files occupy most of the space.

Care needs to be taken to separate out concepts that are popularly associated with power laws, e.g., *scale invariant*, which are a property of the distribution, not the generating process. The process generating data fitted by a power law can be remarkably random, e.g., the length of words in text produced by monkeys typing.<sup>323</sup>

## 9.4 Markov chains

A *finite state machine* (FSM) is a machine represented by a set of distinct states, connected by edges denoting the possible transitions that can occur when a given event occurs, such as when a particular character is input (FSMs are deterministic).

A *Markov chain* (MC) is also a machine represented by a set of distinct states connected by edges, but the possible transition is chosen at random based on the transition probability of each edge (the transition probabilities, out of any state, that is not an absorbing state, add to one); the next state only depends on the current state, i.e., the system is memoryless.

A Markov chain is a *discrete-time Markov chain* (DTMC), if the transition between states occurs at fixed time intervals; if the time interval between state transitions is not fixed, the Markov chain is a *continuous-time Markov chain* (CTMC) (the memoryless requirement means that transition times must have an exponential distribution). If the transition time

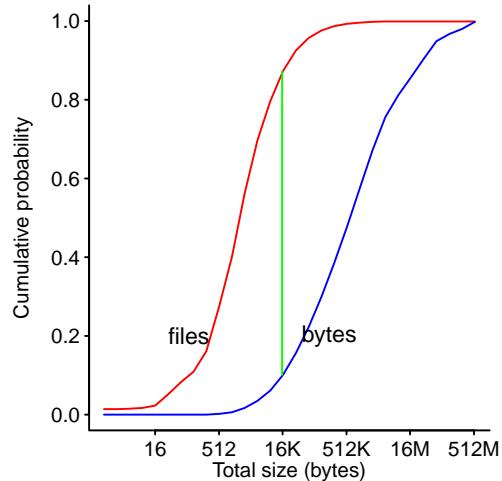


Figure 9.18: Cumulative probability distribution of files size (red) and of number of bytes occupied in a file system (blue). Data from Irlam.<sup>756</sup> code

<sup>vi</sup> The term *sub-exponential* is sometimes used to describe tails that decay slower than exponential and *super-exponential* for tails that decay faster than exponential.

depends on how long the system has been in the current state, it is a *semi-Markov process* (SMP).

Finite state machines provide a useful abstraction for modeling user interfaces. A study by Oladimeji<sup>1185</sup> investigated the user interface of the Alaris volumetric infusion pump (a medical device used for controlled automatic delivery of fluid medication, or blood transfusion, to patients); the user interface includes 14 buttons and an LCD display. Figure 9.19 shows the available transitions between states.

A FSM can be represented as a control flow graph. By using this representation functions in the `igraph` package can be used to answer questions such as: the maximum number of button presses needed to get to any state (12; the `path.length.hist` function returns a count of all possible path lengths) and the average number of presses to transition between any two states (4; using the `average.path.length` function).

If the behavior of a system (that can be represented using a FSM) is monitored, the probability of occurrence of every transition between states can be calculated. If the behavior represents typical user interaction with the system, then the probabilities can be used to create a Markov chain for this typical behavior.

A study by Tarasov, Mudrankit, Buik, Shilane, Kuenning and Zadok<sup>1537</sup> used data on the lifetime of source files in various systems, such as the Linux kernel, to generate realistic filesystem contents (for deduplication analysis). Figure 9.20 shows a Markov chain representing the life of the source files in the Linux kernel (from being Initialised to new, through modified/unmodified to deleted and reaching the Terminal state). The measurement snapshot occurred at each of the 40 releases between versions 2.6.0 and 2.6.39, with an average of 23k files per snapshot; the time between releases is roughly constant, so this might be considered a discrete-time Markov chain.

The `graph.data.frame` function assumes there is a link between the row values in two columns (from and to vertices) and builds a graph based on this assumption. The `V` and `E` functions access the vertices and edges of the graph and various attributes can be set and may be subsequently used by `plot`.

```
library("igraph")

atc=read.csv(paste0(ESEUR_dir, "probability/atc12-gra.csv.xz"), as.is=TRUE)
atc_gra=graph.data.frame(atc, directed=TRUE)

V(atc_gra)$frame.color=NA
V(atc_gra)$size=12 ; V(atc_gra)$color="yellow"
E(atc_gra)$arrow.size=0.5 ; E(atc_gra)$color="red"
E(atc_gra)$weight=E(atc_gra)$linux
E(atc_gra)$label=E(atc_gra)$weight/100

# layout.lgl outperforms the default layout for this graph
plot(atc_gra, edge.width=0.3*sqrt(E(atc_gra)$weight),
     edge.curved=TRUE, layout=layout.lgl)
```

The algorithm used by `plot` to layout a graph makes use of randomization, which means that the layout returned by every call is different.

### 9.4.1 A Markov chain example

A study by Perugupalli<sup>607,1238</sup> investigated the reliability of gcc based on the reliability of its major subsystems. Information on the probability of a subsystem experiencing a failure was calculated using the regression suite for gcc version 3.3.3 (which contains tests for 110 faults present in gcc version 3.2.3, out of 2,126 tests, of which 55 were traced back to the source code of a single subsystem; the others faults involved multiple subsystems). The researchers did not attempt to analyse failures involving more than one subsystem and assumed that subsystems fail independently of each other.

Subsystems were identified by instrumenting gcc to count the number of calls between pairs of functions, made while executing the regression suite (this is not actually Markov chain-like behavior because the called functions return, which is not transition-like behavior). The 1,759 traced functions were manually assigned to one of 13 internal subsystems (e.g., parsing, tree optimization and register allocation),

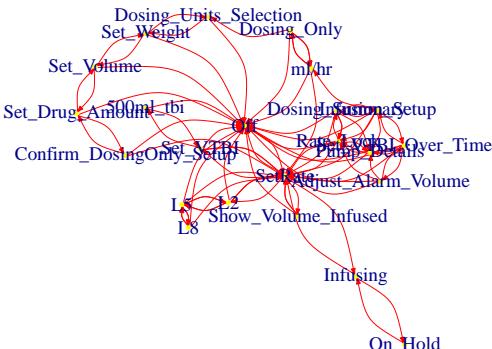


Figure 9.19: Graph of available state transitions for Alaris volumetric infusion pump (the button presses that cause transitions between states are not shown). Data kindly supplied by Oladimeji.<sup>1185</sup> [code](#)

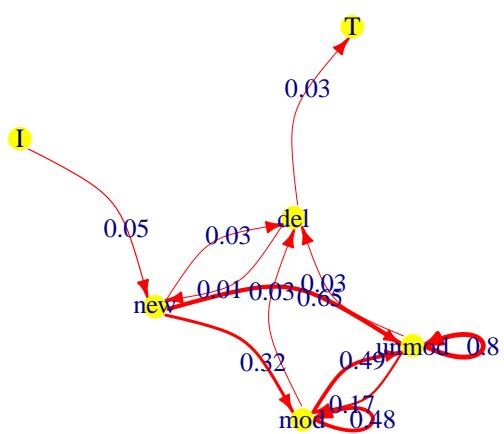


Figure 9.20: Discrete-time Markov chain for created/modifed/deleted status of Linux kernel files at each major release from versions 2.6.0 to 2.6.39. Data from Tarasov.<sup>1537</sup> [code](#)

The reliability of gcc version 3.2.3 might be estimated using:

$$R = 1 - \frac{F_c}{T_c} = 1 - \frac{110}{2126} = 0.948$$

where  $F_c$  is the number of source files that it did not correctly compile and  $T_c$  is the total number of files compiled.<sup>vii</sup>

This approach has the advantage of being simple to calculate, but does not provide any information on the impact of individual subsystems on overall reliability, for instance, what is the sensitivity of overall system reliability to behavioral changes to one subsystem?

The probability of reaching subsystem  $n$  from subsystem 1 after  $k$  transitions is  $Q^k$  (where  $Q$  is the matrix of transition probabilities). Summing over all transitions (using an infinite upper bound for the total number of transitions simplifies the mathematics) we get:<sup>1563</sup>

$$S = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1}$$

where:  $I$  is the identity matrix. The expression  $(I - Q)^{-1}$  is easily calculated (i.e., inverting the result of a matrix subtraction).

The matrix  $S$ , is known as the *fundamental matrix* and can be used to calculate a variety of properties of systems modeled by the Markov chain.

The composite and hierarchical methods are two techniques for combining information on subsystem usage (i.e., subsystem transition probabilities and subsystem reliability, calculated using the above formula) to calculate the reliability of a complete system:

- composite method:<sup>287</sup> this calculates the probability of a successful transition between each subsystem, by multiplying the transition probabilities of each subsystem by the probability of the subsystem executing successfully. These individual successful transition probabilities are used to calculate the successful transition probability from the initial subsystem to the final subsystem (i.e., the system's fundamental matrix). The estimated reliability calculated for gcc is 0.9972,<sup>viii</sup> (see [reliability/gcc-reliability.R](#)).
- hierarchical method: if  $R_i$  is the reliability of a subsystem, the probability of all executions of that subsystem being successful is  $R_i^{N_i}$ , where  $N_i$  is the number of transitions to subsystem  $i$  during one execution of the system. Assuming that subsystems fail independently, the expected value of the system reliability is:

$$R = E \left[ \prod_{i=1}^n R_i^{N_i} \right]$$

Assuming subsystems are highly reliable and the variance in the number of subsystem transitions is very small, the first order Taylor approximation can be used:

$$R \simeq \prod_{i=1}^n R_i^{V_i}$$

where:  $V_i = E[N_i]$  is the expected number of times a transition occurs to subsystem  $i$ , during a single execution of the complete system;  $V_i$  is obtained by solving:

$$V_i = q_i + \sum_{j=1}^n V_j p_{ji}$$

where  $q_i$  is the probability that execution starts with subsystem  $i$  and the  $p_{ji}$  are obtained from the subsystem transition probability matrix (see [reliability/gcc-reliability.R](#)).

The `markovchain` package supports discrete time Markov chains and the `msm` package supports continuous time through the use of multi-state models.

---

<sup>vii</sup> The calculated reliability is very low because it is based on compiling a test suite of short code samples designed to reveal faults.

<sup>viii</sup> If the 55 fault count used in this analysis is plugged into the simple formula used above, the reliability estimate is 0.974.

## 9.5 Social network analysis

The popularity of web based social networks has made the mathematics of social network analysis a fashionable research topic. Unfortunately many published papers involve little more than claiming to have found a power law, with only pretty pictures and hand waving to show. An example of the kind of descriptive statistics encountered in a social network analysis appears in Table 7.1; the section on regression modeling discusses building models based on network data.

Social networks are represented as graphs and the `igraph` package supports reading many graph data representation formats, along with a wide range of operations and analysis on graphs.

A study by Canfora, Cerulo, Cimitile and Di Penta<sup>250</sup> analysed the developer's mailing lists for FreeBSD and OpenBSD, to obtain information on what they called *Cross-System-Bug-Fixings*; the data contains information on 861 unique developers sending email and 1,062 unique developers receiving email. Both FreeBSD and OpenBSD were forked from a common base and not only continue to share common code but faults fixed in one are often applied, some time later, to the other. Figure 9.21 was produced using code very similar to that used for the Markov chains in Figure 9.20.

Many real world collections of linked node contain subgroups (e.g., clusters of developers or related code modules), and there are a variety of algorithms for detecting these subgroups. Care needs to be exercised in interpreting the clusters returned by these algorithms, as there may be many distinct high-scoring solutions and a clear global maximum may not exist.<sup>592</sup>

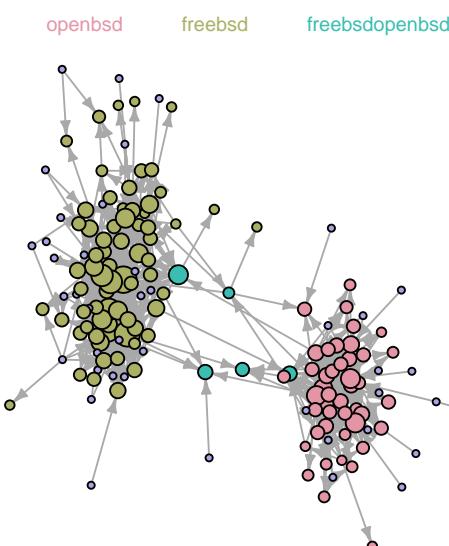


Figure 9.21: Directed graph of emails between FreeBSD and OpenBSD developers, plus a few people involved in both discussions, with developers who sent/received less than four emails removed. Data from Canfora et al.<sup>250</sup>  
code

## 9.6 Combinatorics

The analysis of some systems makes it necessary to consider combinations of various items, and there is a need to enumerate all possible sequences, to find the total number of different sequences of items that could occur (or other related questions). The mathematics used to solve this kind of problem is known as *combinatorics*.

A few of the functions frequently used in combinatorial problem solving are included in R's base system, including:

- the `choose` function takes two arguments,  $n$  and  $k$  and returns the value  $\frac{n!}{k!(n-k)!}$ , often written as  $\binom{n}{k}$ ; the number of ways of selecting  $k$  items from  $n$  items,
- the `combn` function takes two arguments,  $x$  and  $k$  and returns an array containing all combinations of the elements of  $x$  taken  $k$  at a time.

When an item is drawn, with replacement, from a pool of items the probability of drawing the same item again is unchanged, when drawing without replacement the probability will decrease by the appropriate amount. An item is distinct if it is treated as being different from all other items in the pool (even when drawing with replacement), e.g., there are four items in the pool `x=c("a", "a", "b", "c")`, but only three of them are distinct.

Table 9.1 show how the `iterpc` function in the `iterpc` package can be used to generate sequences based on the distinctness of the items and whether they are drawn with replacement or not.

		Distinct	
		True	False
Replacement	True	<code>I=iterpc(5, 2, replace=TRUE)</code>	<code>x=c("a", "a", "b", "c") I=iterpc(table(x), 2, replace=TRUE)</code>
	False	<code>I=iterpc(5, 2)</code>	<code>x=c("a", "a", "b", "c") I=iterpc(table(x), 2)</code>

Table 9.1: Example `iterpc` calls generating particular kinds of sequences of length two (by passing the value returned to `getall`, e.g., `getall(I)`).

The treatment of item ordering is another factor, when considering all possible permutations; is the ordering of items significant or not, e.g., are the sequences `a,b` and `b,a` treated as different or equivalent? When the ordering of items is significant calls to `iterpc` need to set the optional argument `ordered` to `TRUE`, e.g., `I=iterpc(5, 2, ordered=TRUE)`.

### 9.6.1 A combinatorial example

This example illustrates the kind of detailed analysis needed to solve a practical combinatorial problem.

A study by Jones<sup>[789](#)</sup> investigated developer preferences for ordering members within C `struct` types. The hypothesis was that members having the same type are likely to be grouped together within the same `struct` type.

The data contains enough instances of `struct` types containing between three and eight members, for the sample to be analysed with a reasonable level of confidence.<sup>[ix](#)</sup>

If a `struct` contains  $n$  members, the number of possible member sequences is  $n!$ . However, we are only interested in member types and don't care about permutations of members having the same type. The number of different member type sequences is  $\frac{n!}{n_1!n_2!\dots}$  where  $n = n_1 + n_2 + \dots$  and  $n_1, n_2$ , etc are the number of members having a given unique type.

Taking the example of a `struct` containing four members, two of type `x` and two of type `y` the possible sequences of member types within a `struct` type are:

`xxxy` `xyxy` `yxxx` `xyyx` `yyxy` `yyxx`

and if two members are of type `x`, one of type `y` and one of type `z`, the possible member type sequences are:

`xxyz` `xxzy` `xyxz` `xyzx` `xzxy` `xyyx` `yxzx` `yzxx` `zxyx` `zxyy` `zyxx`

In the first case members are grouped together in  $\frac{1}{3}$  of cases and in the second, in  $\frac{1}{2}$  of cases.

If there are  $t$  different types, there are  $t!$  possible unique sequences of types. If the ordering of `struct` members is random, the probability of encountering a definition in which all members having the same type are grouped together is:

$$\frac{t!}{\frac{n!}{n_1!n_2!\dots n_t!}}$$

For the two examples above the probabilities of encountering a member ordering, where identical types are grouped together, are:  $\frac{2!}{4!} = \frac{1}{12}$  and  $\frac{3!}{2!2!} = \frac{3}{12}$  (which is already known from enumerating out all possible sequences).

When a `struct` contains four members, as in the above examples, it is not possible to distinguish between a developer intentionally choosing an order and random selection. For `structs` types containing five members, the probability of random selection of member order, grouping together the same member types is surprisingly high; see the fifth column in Table 9.2.

Total members	Type sequence	structs seen	Grouped occurrences	Random probability	Occurrence probability
4	1 1 2	239	185	0.50	$2.83 \times 10^{-18}$
4	1 3	185	146	0.50	$4.75 \times 10^{-16}$
4	2 2	98	61	0.33	$4.58 \times 10^{-9}$
5	1 1 1 2	57	50	0.40	$1.03 \times 10^{-13}$
5	1 1 3	94	61	0.30	$3.13 \times 10^{-12}$
5	1 2 2	86	49	0.20	$5.18 \times 10^{-14}$

Table 9.2: Various forms of `struct` types containing a given number of members, one possible type grouping, number of actual `struct` types measured, number having grouping, probability that one type will contain this grouping and probability that the number grouped, out of total seen, will be so grouped. Data from Jones.<sup>[789](#)</sup> [code](#)

<sup>ix</sup> The number of `struct` types containing a given number of members decreases approximately logarithmically with increasing number of members,<sup>[785](#)</sup> i.e., most member sequences are relatively short.

Table 9.2 shows source code measurement counts and calculated probabilities for struct types containing four and five members: the column *Total members* lists the number of members in the type, *Type sequence* is a possible grouping of member types for a given number of member types, *structs seen* is the number of measured structs containing the given number of members/types, *Grouped occurrences* is the number of measured structs having the grouping listed in the first column, *Random probability* is the probability of this grouping occurring randomly in one struct declaration containing the given number of members and types, *Occurrence probability* is the probability of *Grouped occurrences* out of *structs seen* occurring, when the probability of a single instance occurring is *Random probability*.

This analysis shows it is not possible to confidently distinguish between random and intentional ordering, for individual struct types. However, programs contain many such type definitions, and if we label each one "Yes" or "No", depending on whether their member types are grouped or not, this list of Yes/No labels has a binomial distribution, and the probability of the given number of Yes/No labels occurring through chance can be calculated.

Taking the example of a struct containing four members, two of type x and two of type y, the sample analyzed here contains 98 such types, with 61 of them having grouped member types (see columns 3 and 4 of Table 9.2). The probability of this occurring, when the probability of a single (random) occurrence is  $\frac{1}{3}$ , is calculated using `pbinom(61-1, 98, 1/3, lower.tail=FALSE)`, whose value is `4.58272e-09` (the `lower.tail=FALSE` option is used because we are interested in the probability of seeing 60 or more occurrences).

Figure 9.22 shows the measured percentage of struct types whose members are grouped by type (red pluses) and the percentage that would occur with random ordering (blue line). The green line is the 99.9% probability bound, for the likelihood that 100 structs, all sharing the same member types, will all have their members grouped by type when member ordering is chosen at random. The distance of the red crosses from the 99.9% bound shows that grouping of members by type is very unlikely to have been driven by random selection.

## 9.6.2 Generating functions

Generating functions are discussed here purely to inform readers about a powerful technique, which is significantly different from the traditional approach to solving probability problems using factorials; this technique is capable of solving problems that appear to be otherwise intractable. If it is not possible to derive an expression specifying how many possibilities can occur in some situation, then a search for the appropriate generating function may provide an answer.

Generating functions are starting to be covered in texts on probability; some mathematical sophistication is required.

A generating function is a polynomial  $a_0x^0 + a_1x^1 + \dots + a_nx^n$ , where the coefficients  $a_n$  encode information about the quantity of interest.

The following is a simple example that could just as easily be calculated using factorials, but illustrates the idea. How many ways can five items be selected, if A can be selected 0 or 1 times, B can be selected 0, 1 or 2 times and C can be selected 0, 1, 2, 3 or 4 times? The generating function is (see the suggested reading for why this works):

$$(1+x)(1+x+x^2)(1+x+x^2+x^3+x^4) = x^7 + 3x^6 + 5x^5 + 6x^4 + 6x^3 + 5x^2 + 3x + 1$$

the coefficient of  $x^5$  is 5, so five different items orderings are possible.

A more complicated example is when items have a particular value and sequences that sum to a specific total are required. If A is worth 1, B is worth 3 and C is worth 5, the generating function is:

$$(1+x+x^2+\dots)(1+x^3+x^6+\dots)(1+x^5+x^{10}+\dots) = 7x^{11} + 7x^{10} + 6x^9 + 5x^8 + 4x^7 + 4x^6 + 3x^5 + 2x^4 + 2x^3 + x^2 + x + 1$$

the coefficient of  $x^{10}$  is 7, so there are seven different ways of selecting items that sum to ten.

The `polynom` package supports the symbolic manipulation of polynomials.

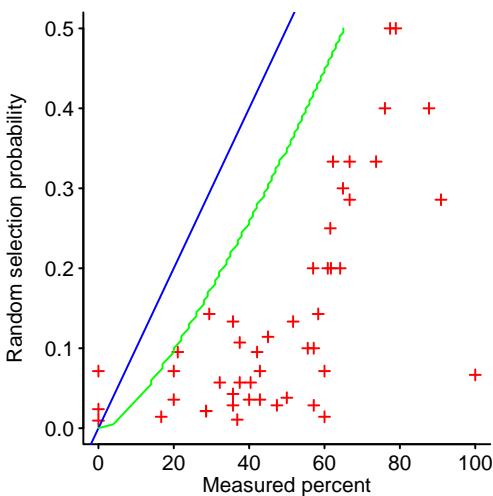


Figure 9.22: Expected probability of a single instance (y-axis) against the probability of a measured struct type having grouped member types (x-axis); when both probabilities are the same points will be along the blue line.  
Data from Jones.<sup>789</sup> code

# Chapter 10

## Statistics for software engineering

### 10.1 Introduction

The output from statistical analysis should be treated as a guide, not as definitive.

Correlation does not imply causation, a common mantra that is always worth repeating.

The developer input to the data analysis process is to use their domain knowledge to guide the search for patterns of behavior, and to provide one or more interpretations to any patterns that are uncovered.

The results from data analysis may vary with the person doing the analysis;<sup>1439</sup> for instance, people may use a technique because it is the one they know how to use, rather than the technique best suited to the data being analyzed.

Traditionally, statistical techniques have had to be practical to perform manually. This has resulted in general statistical problems being split into a profusion of specific subproblems and the creation of techniques tailored to handle each case. Doing statistic analysis this way, involved mapping the sample characteristics to a particular subproblem and then applying the corresponding technique. Computer availability makes it practical to apply general solution techniques and general, powerful and robust statistical techniques are available.<sup>450</sup> however, many existing users of statistical techniques have simply switched from manual to computer based calculation of the familiar historical techniques, without appreciating the original design rational for these techniques. Many of the statistical techniques appearing in this book are impractical to apply manually (e.g., the bootstrap) a computer is required.

Existing books often invest effort massaging data into a form that permits the use of techniques that depend on the data having a Normal distribution (also known as a \_Gaussian distribution<sup>i</sup>). The reasons for this are historical (assuming Normality made the analysis tractable in the days before computers) and data in the Social sciences (early adopters of statistical techniques and a major market for statistical books) appearing to be drawn from a Normal distribution (despite the claims made, data in this field often does not have a Normal distribution<sup>1073</sup>). It could be said that nobody ever got fired for assuming a Normal distribution.

Measurements of software engineering processes often produce values that are not drawn from a Normal distribution; the Exponential and Poisson distributions are relatively common; measurement samples that are best described by a Normal distribution do occur, but they do not have the dominant market share encountered in other, non-software related, domains (e.g., the social sciences).

The input to statistical analysis is a sample and usually some expectations of behavior; the expectations may be explicit (e.g., measurements are independent of each other) or implicit (e.g., the choice of a statistical technique that only produces reliable results for samples drawn from a population having certain characteristics).

The possibility for detecting patterns that might be present in a sample depends on the quality and quantity of the measurement data:

---

<sup>i</sup> This book uses the term Normal because it appears to be more widely used.

- quality: noise in the measurement process and errors in post measurement processing (e.g., incorrect conversion of file formats or inaccurate calculations of values derives from the raw data) are some of the problems that affect data quality,
- quantity: the number of measurements impacts the power and significance of statistical tests, and the confidence bounds on the results of statistical analysis.

Finding a pattern in the data having the desired level of statistical certainty, moves the discussion on to the practical engineering consequences of what has been found, e.g., mountain or molehill. A discussion of practical engineering consequences of patterns is outside the scope of this book.

### 10.1.1 Statistical inference

The most commonly used statistical inference technique makes use of *frequentist* methods, i.e., how often events occur and long-run averages. All techniques have problems associated with their use and frequentist, being the most widely used, has the greatest number of detractors; a common problem is misuse of the concept of p-value; any widely used technique will have a common failure mode, simply because of the varying skills of the people using it. The p-value is the fall-guy of the frequentist approach to statistical analysis.

The frequentist approach is the technique predominantly used in this book because it is commonly used in statistical books and articles; it is used by most of the R packages and readers are likely to encounter it when interacting with other people involved in analysing and using data.

Another technique is *Bayesian statistics*, which is growing in popularity; some R packages use this approach internally. A Bayesian approach has the potential to extract more information from data, by making use of information about prior beliefs. What is known as the *prior*, is a reasonable value for the probability of the event occurring, estimated prior to any measurements being made (the measurements get factored in later); the selection of a suitable prior opens the door to the bias of opinion and policy guidelines,<sup>191</sup> e.g., a Bayesian approach to deciding whether the accused is guilty runs into the problem that many legal systems assume people are innocent until proven guilty (i.e., the prior is zero), a belief that percolates through calculations to always produce a not-guilty result.

A study by Furia<sup>531</sup> reanalyzes several software engineering datasets using Bayesian techniques.

*Maximum likelihood estimation*, MLE, is a technique that finds the set of parameters for a model that make the observed data most likely to have occurred. It has been said that academics like maximum likelihood estimation because it is a way of showing off their calculus within the statistics syllabus...

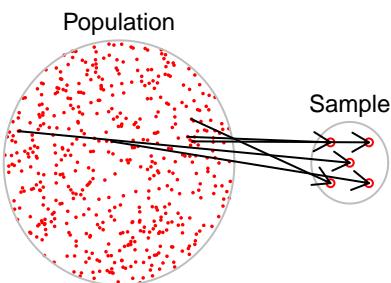


Figure 10.1: Example of a sample drawn from a population. [code](#)

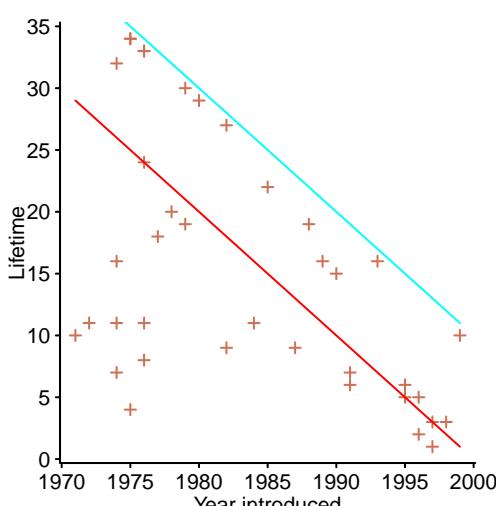


Figure 10.2: Date of introduction of a cpu against its commercial lifetime; processors ceasing production in 2000 or 2010 would appear along one of the lines. Data from Culver.<sup>343</sup> [code](#)

## 10.2 Samples and populations

It may not be practical to measure every member of a population and the subset of the population measured is known as a *sample*; see Figure 10.1.<sup>ii</sup> Depending on the question being asked, a set of measurements may be a population or a sample. For instance, measurements of one particular program yields the parameters of a population when the questions being asked concern just that one program, but they become the statistics of a sample when generalizing the findings to questions about other programs (including future versions of the one measured).

A sample is selected as a proxy of the entire population. There are a variety of sampling techniques, including:

<sup>ii</sup> The term *statistic* applies to values calculated from a sample, while the term *parameter* applies to values calculated from a population. In some equations the value  $N - 1$  is used, when  $N$  might appear to be more appropriate. A mathematical distinction occurs between samples and populations, in that sample estimates are based on degrees of freedom of the sample, i.e., the number of members of the sample minus one, while population parameters are based on the number of members of the population, i.e.,  $N$ .

- a *survey sample* is collected when the items to be measured (often via a questionnaire) are selected from a population assumed to share (unknown) fixed characteristics. The measured characteristics of the random sample, drawn from this fixed population, are used to estimate the characteristics of the population. The analyse of a sample obtained via a survey is *design-based* (rather than *model-based*). See Section 13.5 for a discussion of questionnaire based surveys,
- a *prospective study* collects data as events unfold. Figure 10.2 shows the date of introduction of a cpu against its commercial lifetime, in years.<sup>343</sup> Processors that ceased production in 2000 or 2010 would appear along one of the two colored lines,<sup>iii</sup>
- a *retrospective study* collects data after events have taken place,
- a *convenience sample*, as its name implies, makes do with what is available,
- *snowball sampling*, or *chain sampling* starts with an initial list of subjects, who are asked to propose other subjects whom to them, with the process iterating until the number of new subjects falls below some threshold,
- stratified sampling divides the population into what are known as *strata*, with the strata chosen such that similar cases tend to cluster within each one; each of these strata are then sampled (using, say, random sampling) to produce the final sample (which is a set of distinct stratum, see Figure 10.3),
- sequential sampling is covered in Section 13.2.4,
- interval sampling divides the measurement interval into a series of fixed points and samples at just these points. The width of the sampling intervals puts a lower bound on the behavior that can be resolved. An experimental study<sup>iv</sup> by Kistowski, Block, Beckett, Lange, Arnold and Kounev<sup>855</sup> measured power consumption, using programs from SPEC's Server Efficiency Rating Tool, at load level increments of 2% (crosses) and 10% (lines); see Figure 10.4. A cost/benefit analysis would compare the greater accuracy obtained using finer measurement intervals against the likelihood of sudden jumps in the value of the response, that could have a noticeable impact on the results.

Occasionally the subjects of interest are not present in the sample. For instance, the damage experienced by aircraft returning from combat, during the second world war, was analysed with a view to improving aircraft survival rate. One of the statisticians involved pointed out that important subjects were missing from the sample,<sup>1013</sup> aircraft that had not returned. The return of a damaged aircraft provides evidence that the damaged areas are not critical to survival; it was those areas not damaged in returning aircraft that are likely to be critical to survival.

Guy<sup>645</sup> proposed a *strong law of small numbers*, "There aren't enough small numbers to meet the many demands made of them.", listing 35 examples of numeric patterns found in samples calculated using small integer values that disappear when larger integer values are used. The greater number of large values reduces the likelihood of coincidental correctness.

Figure 10.5 shows the four connected statistical characteristics of a sample; given the values of three of them, the fourth can be calculated.

While gathering a representative sample of the population as a whole is a common requirement, sometimes samples having other characteristics are of interest, e.g., being representative of diversity.<sup>1128</sup>

### 10.2.1 Effect-size

*Effect-size* is the degree to which the characteristic of interest is present in the population (from which a sample is drawn), e.g., if we are interested in the difference in the performance of developers before and after attending a training course, how big is the difference (answering this question may be the reason for obtaining measurements)?

The question to ask about a calculated effect-size is: "Does it matter?" The larger the effect-size the more likely it is to be of interest in practice; in some cases a small effect-size may

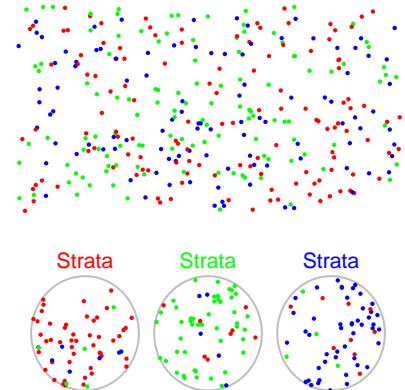


Figure 10.3: A population of items having one of three colors, along with samples of the three strata (imperfect item selection introduces noise in the samples). [code](#)

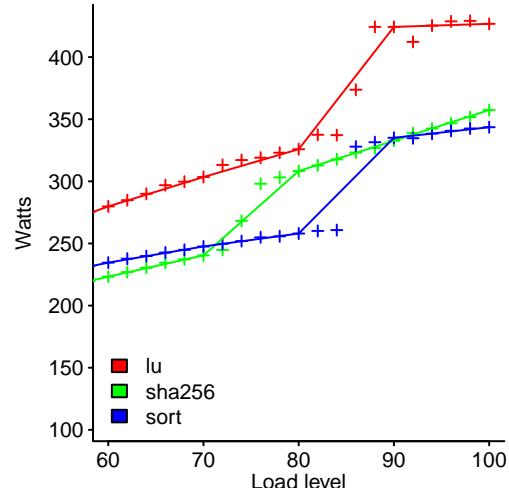


Figure 10.4: Power consumed by three SERT benchmark programs at various levels of system load; crosses at 2% load intervals, lines based on 10% load intervals. Data kindly provided by Kistowski.<sup>855</sup> [code](#)

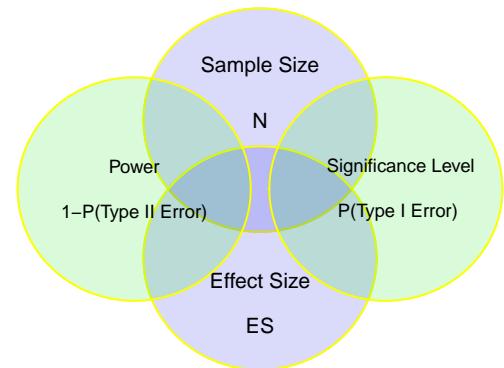


Figure 10.5: The four related quantities in the design of experiments; given three, the fourth can be calculated. [code](#)

<sup>iii</sup> Email discussion with the author confirmed that the data had not been updated since 2010.

<sup>iv</sup> The study was experimental because it did not meet all the requirements for an official SERT run.

be of interest (e.g., a small difference multiplied over a large population can have a large impact), while in other cases only a large effect-size is of interest (e.g., when the population is small a large effect-size may be needed to have a large impact).<sup>v</sup>

Smaller effect-sizes are likely to be more costly to detect because more measurements are needed to isolate small effect-sizes compared to larger ones.

Figure 10.6 shows how percentage differences in the presence of a condition in a population can have a dramatic effect on the false positive rate (in red), for the same statistical power and p-value.

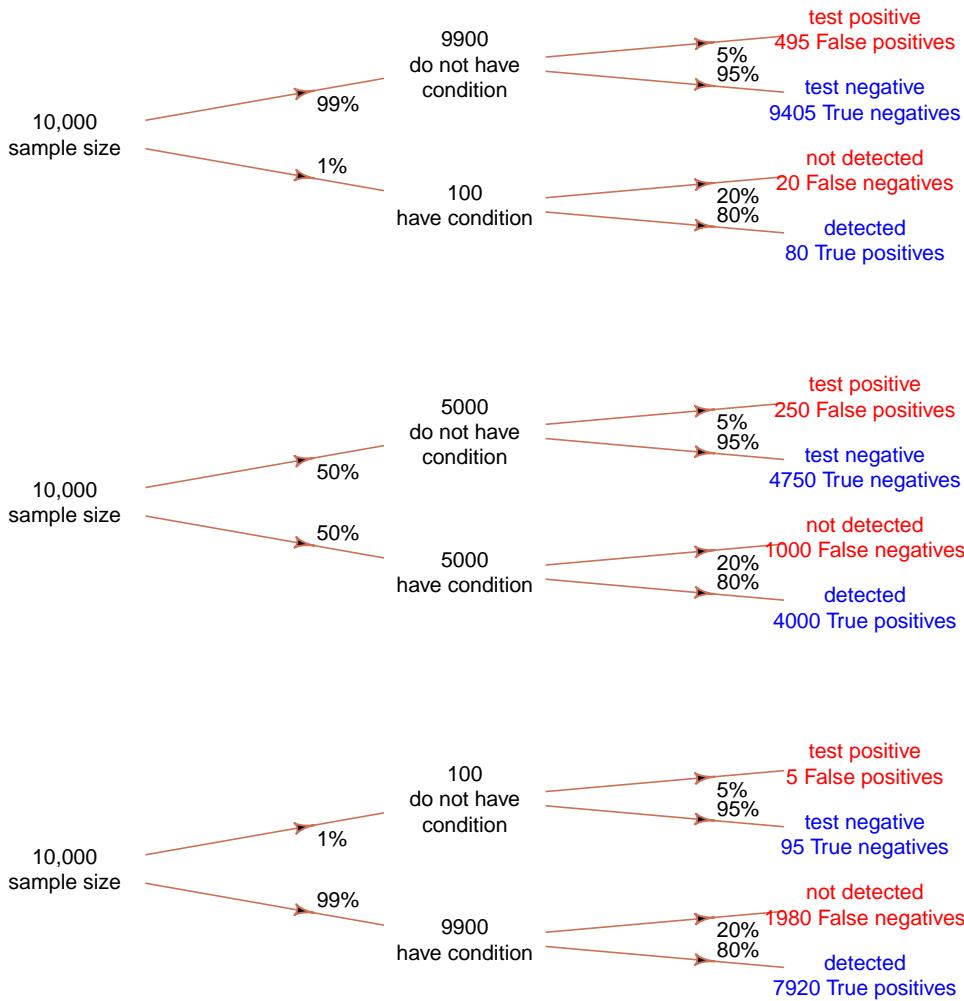


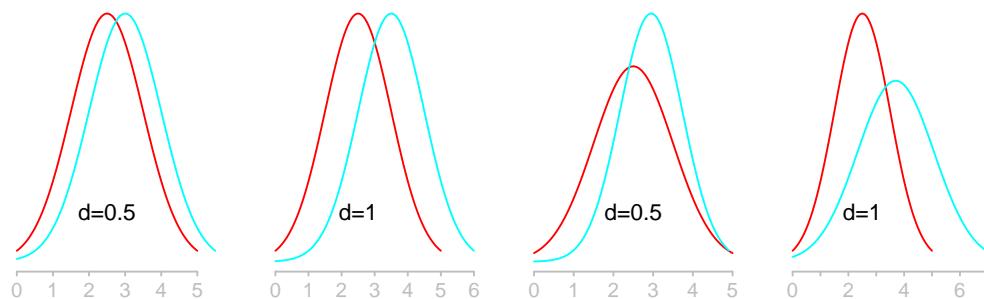
Figure 10.6: Examples of the impact of population prevalence, statistical power and p-value on number of false positives and false negatives. [code](#)

Methods for calculating effect-size depend on the kind of analysis being performed on the sample,<sup>444</sup> and include the following:

- correlation, e.g., the Pearson correlation coefficient, is a measure of effect-size,
- combining information on the mean and standard deviation of two samples into a single value. For instance, Cohen's  $d$  is one measure used when samples have similar standard deviations and is given by (other approaches adjust the calculation of the standard deviation):  $d = \frac{\mu_1 - \mu_2}{\sigma_{pooled}}$ . There are a variety of effect-size calculations associated with Cohen's name.

Figure 10.7 illustrates how differences in mean and standard deviation, of two distributions, result in a given Cohen's  $d$ ,

<sup>v</sup> Statistical books<sup>309</sup> and papers sometimes concern themselves with questions of where to draw the lines that delimit large/medium/small effect-sizes, an approach that might be applicable when researchers are more interested in publishing papers than making useful discoveries.



- odds ratio (i.e.,  $odds = \frac{p}{1-p}$ ), that is the ratio of the odds of an event occurring in one sample divided by the ratio of the same event occurring in the other sample (perhaps a control group),

### 10.2.2 Sampling error

If the reader agrees that sampling error is an important issue, this section can be skipped. Otherwise, read on and be frightened into agreeing.

The Central Limit theorem is a statement about the mean value of samples drawn from a population. If the population has a finite variance (power laws with an exponent between zero and two have an infinite variance), then the distribution of sample means converges to a Normal distribution as the sample size,  $N$ , increases (it does not matter which distribution the population has, it is the distribution of the sample means that converges to the Normal).

How quickly does the distribution of sample means converge? The Berry-Esseen theorem gives the best known estimate of the convergence of the distribution of the mean of independent, identically distributed, variables to a Normal distribution:

$$|F_n(x) - \Phi(x)| \leq \frac{0.34(\rho + 0.43\sigma^3)}{\sigma^3\sqrt{N}}$$

where:  $F_n$  is the cumulative distribution function of the means,  $\Phi$  the cumulative distribution function of a Normal distribution,  $\rho$  the third moment of  $x$  (and less than infinity),  $\sigma$  the standard deviation and  $N$  the sample size.

The only parameter available for influencing the error is the number of measurements; the error is proportional to:  $\frac{1}{\sqrt{N}}$ , e.g., to halve the error in the sample mean, the sample size needs to increase by a factor of four.

Figure 10.8 shows the distribution of mean values for samples drawn from three different distributions (using two sample sizes); the vertical lines are 95% confidence bounds.<sup>vi</sup>

A study by Chen, Chen, Guo, Temam, Wu and Hu<sup>281</sup> measured the performance of programs in the SPEC CPU2006 benchmark using 1,000 sets of input data for each program. As an exercise in sampling let's assume we only have access to three of the possible 1,000 input datasets, what range of execution times might we expect to see from processing just three datasets?

Figure 10.9 was obtained by randomly sampling three items from the population of 1,000 and repeating the process 100 times. The red cross is the sample mean and the vertical brown lines each sample's standard deviation; the blue line is the mean for the population of 1,000 input sets and the green lines the bounds of this population's standard deviation.

Figure 10.10 shows the distribution of sample means for sample sizes of 3 and 12 items. As expected, the larger samples show less variation in the mean value.

Sources of noise (i.e., random variability) in a sample include the following:

- measurement error caused by imperfect tools used to make measurements, which can include coding mistakes,
- demographic variability, e.g., measurements of particular kinds of programs, or developers working in a single location or for one company,

Figure 10.7: Visualization of Cohen's  $d$  for two normal distributions having different means and the same standard deviation (two left), and different mean and standard deviations (two right). Exponential Exponential size=10 size=20

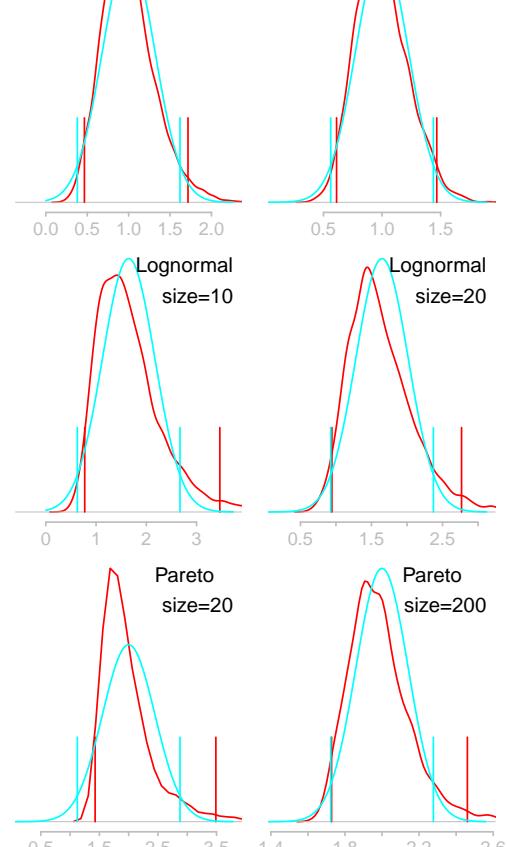


Figure 10.8: Distribution of 4,000 sample means, for two sample sizes, drawn from exponential (upper), lognormal (center) and Pareto (lower) distributions, vertical lines are 95% confidence bounds. The blue curve is the Normal distribution, predicted by theory. code

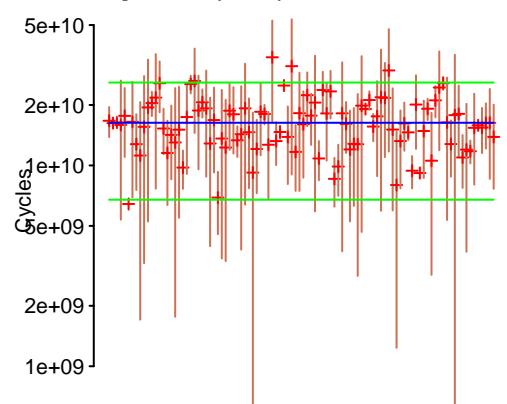


Figure 10.9: Mean (red) and standard deviation (brown lines; they are not symmetrical because of the log scaling) of samples of 3 items drawn from a population of 1,000 items (blue line mean, green line standard deviation). Data kindly provided by Chen.<sup>281</sup> code

<sup>vi</sup> There will be random fluctuations from the random draw of the sample.

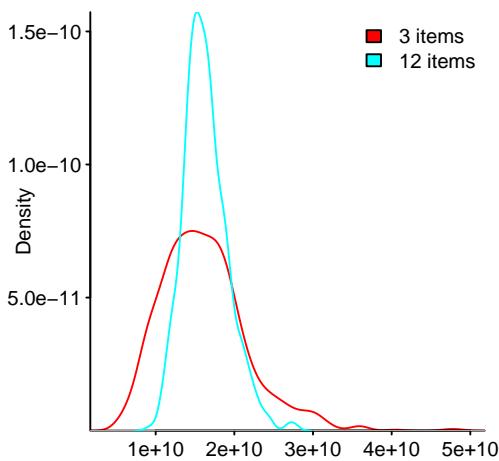


Figure 10.10: Density plot of mean of samples containing 3 or 12 items randomly selected from a data set of 1,000 items; process repeated 1,000 times for each sample size. Data kindly provided by Chen.<sup>281</sup> [code](#)

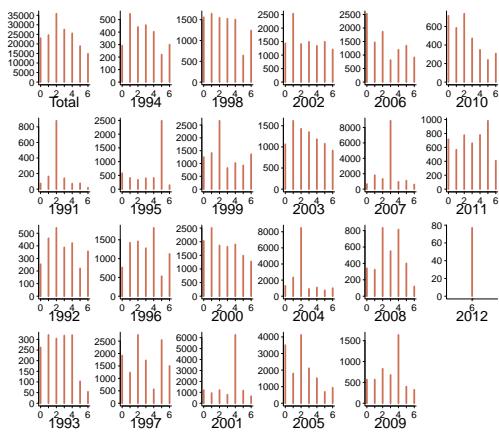


Figure 10.11: Number of commits to glibc for each day of the week, for the years from 1991 to 2012. Data from González-Barahona et al.<sup>591</sup> [code](#)

- environmental variability is the sea in which developers swim, or have swum in the past, e.g., company culture or habits acquired from early teachers.

Figure 10.11 shows the number of commits to glibc<sup>591</sup> for each day of the week, separated out by year. The plot in the top left shows daily totals over all years. The combined plot suggests that most commits occur near the middle of the week, with the number falling off towards the beginning and end of the week. However, the yearly plots rarely show anything like this pattern; is any interpretation of the pattern of commits in the combined plot a just-so story?

### 10.2.3 Statistical power

If an effect exists and an experiment is performed to measure it, what is the likelihood that the effect will be detected? The numeric answer to this question is known as *statistical power*, of the experiment. The *power* of a statistical test is its ability to detect a difference when one actually exists in the data. Failing to detect an effect, when one exists, is known as making a *Type II error*, or more commonly as a false negative ( $\beta$  is commonly used to denote the Type II error rate). Techniques for reducing Type II errors include:

- being willing to accept a larger *Type I error*, or more commonly as a true negative ( $\alpha$  is commonly used to denote the Type I error rate),
- sampling from a population thought to have a higher probability of containing the sought after characteristics. For instance, Vasa<sup>1596</sup> excluded releases with less than 30 changed classes in a study of class change dynamics. If a subset of a population is selected to maximise detection rate, care must be taken to ensure that any statement of statistical power refers to the subset population, not the larger population from which it was subselected,
- increasing the number of measurements made, i.e., sample size.

As an example, Figure 10.12 shows the distribution of measurements in two populations: X (red) and Y (green) (e.g., the time taken to execute all possible programs, with all possible input, on two different computers). The upper and middle plot only differ in mean value, while the middle and lower plot only differ in standard deviation. The false positive rate,  $\alpha$ , is shaded in red, and the false negative rate,  $\beta$ , in green. The two rates are connected in that increasing one decreases the other, and vice versa.

When there is a large overlap between samples (middle plot), most of the measurements in either sample have values that suggest they could have drawn from the other sample. In the upper plot, the difference in the sample means makes it more likely that measurements from Y will have values that appear to have been drawn from a different distribution, than samples from X.

The area of the unknown distribution excluding  $\beta$  (i.e.,  $1 - \beta$ ), is known as the power of the test.

A power of 80% is often quoted<sup>309</sup> as being an acceptable lower limit of a test having a high power, just like 5% is often quoted as an acceptable significance level in many contexts.

If there is a need to estimate whether an effect exists (e.g., one computer is faster than another, or a new algorithm uses less memory), before an experiment is run the question to ask is whether a difference (if it exists) is likely to be detected using the available resources (e.g., time and effort needed to obtain a measurement sample). A statistical power calculation shows the tradeoffs that can be made between sample size and probability of detecting an effect (assuming information on population mean, standard deviation and estimated differences between two or more samples).

The `pwr` package supports power analysis calculations for a variety of standard statistical tests. The functions are passed values for three sample characteristics and return the value of the fourth (see Figure 10.5).

A study by Syed, Robinson and Williams<sup>1528</sup> investigated variations in the number of intermittent failures experienced, when using the Firefox browser, at different processor speeds, system memory and hard disc sizes. A total of 11 known coding mistakes, causing intermittent failure (four of these did not produce fault experiences) and nine different hardware

configurations were selected. The conditions expected to cause each mistake to result in a fault being experienced were created, and Firefox was executed 10 times with each hardware configuration. Table 10.1 shows the number of each fault experienced with each hardware configuration.

Mhz-Mb-Gb	124750	380417	410075	396863	494116	264562	332330
667- 128- 2.5	4	10	6	5	2	3	5
667- 256-10	4	8	8	6	4	3	8
667-1000- 2.5	4	7	3	4	3	1	8
1000- 128-10	3	10	3	6	0	1	1
1000- 256- 2.5	3	9	0	6	0	1	2
1000-1000-10	2	9	4	5	0	0	1
2000- 128- 2.5	0	10	5	6	0	0	0
2000- 256-10	2	8	5	7	0	0	0
2000-1000-10	1	7	3	5	0	0	0

Table 10.1: Number of times, out of 10 execution, a known (numbered) coding mistake resulted in a detectable failure of Firefox running on a given hardware configuration (cpu speed-memory-disk size). Data from Syed et al.<sup>1528</sup>

This experiment failed to detect a connection between hardware configuration differences and faults experiences. What is the likelihood that if a connection existed, this experiment would have detected it. Alternatively, how large would the connection need to be for this experiment to detect it?

Analyzing the statistical power of an experiment involving a difference in proportions (i.e., failures before and after) requires an estimate of the effect size (calculated from the proportion of failures before and after a change of hardware specification), the number of runs (10 in this case) and the desired p-value (e.g., 0.05). In this study, there were multiple changes of hardware specification and to keep things simple this analysis calculates the power for one change.

Does a hardware change cause more or fewer faults to be experienced? Without a theory providing a believable rationale for more/less, it has to be assumed that either could occur. In other words, a two-sided test is required.

In the following code: h is the effect size (the ES.h function, from the pwr package, calculates this from the estimated proportion of runs that failed before/after the hardware change), n is the number of runs, sig.level is the p-value significance level and power the statistical power. The argument that is not specified (it is not necessary to specify NULL, this is the default value), or is given a NULL value, is returned by the call.

The default value of the alternative parameter is "two.sided".

```
library("pwr")
```

```
pwr.2p.test(h=ES.h(before, before+diff), n=num_runs, sig.level=0.05, power=NULL)
pwr.2p.test(h=ES.h(before, before+diff), n=NULL, sig.level=0.05, power=0.8)
```

The upper plot in Figure 10.13, shows the power achieved (y-axis), if a given difference in faults experienced does occur (x-axis), the before proportions 0.05, 0.25 and 0.5 are plotted; the power is plotted for 10 and 50 runs.

The probability of a difference being detected from 10 runs is below 0.5 (i.e., less than 50% chance of detecting a difference at a p-value of 0.05 or better), unless a change of hardware has a large impact on the proportion of faults experienced.

The lower plot in Figure 10.13 shows the number of runs needed (y-axis), to have an 80% chance of detecting a given difference (x-axis) in proportion of faults experienced; the before proportions 0.05, 0.25 and 0.5 are plotted, at a significance of 0.05.

This lower plot can be used to find how much difference needs to be experienced for an experiment using 10 runs (per possible fault experience) to be likely to detect it. The failure of this experiment to detect any hardware configuration impact on number of known faults experienced, provides evidence that if any difference does exist, its impact is to add less than 50% or so to the proportion of intermittent fault experiences.

If the pwr package does not contain a function that calculates the power of the statistical test being considered, a Monte Carlo simulation can be used to perform a power calculation

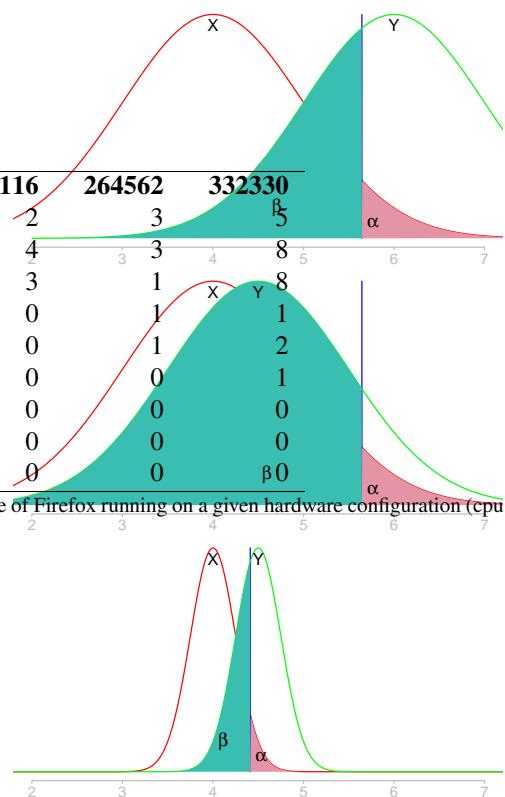


Figure 10.12: The impact of differences in mean and standard deviation on the overlap between two populations ( $\alpha$ : probability of making a false positive error, and  $\beta$ : probability of making a false negative error). [code](#)

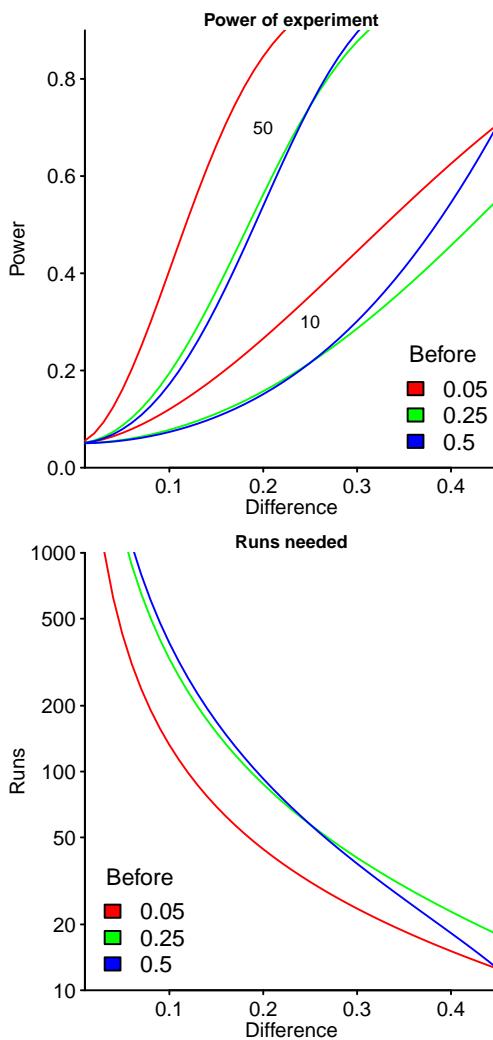


Figure 10.13: Power analysis (50 and 10 runs at various p-values) of detecting a difference between two runs having a binomial distribution (runs needed to achieve power=0.8 at various p-values). [code](#)

for the test being considered. The algorithm simulates an experiment, by obtaining samples from the population(s) that are thought to exist and performing the analysis on each sample, counting each success/failure to detect a difference.

The following code creates two populations and then compares two samples drawn from these populations. The user written function `some_test_statistic` compares two samples and returns the probability that an analysis of two samples will produce a given value; `statistics/boot-power.R` contains an example that checks for a difference in mean value between samples drawn from two populations, see Figure 10.14:

```
boot_power=function(pop_1, pop_2, sample_size, test_stat, alpha=0.05)
{
  num_samples=5000 # Number of times to run the 'experiment'.
  results=sapply(1:num_samples, function(X)
    {
      sample_1=sample(pop_1, size=sample_size, replace=TRUE)
      sample_2=sample(pop_2, size=sample_size, replace=TRUE)
      return(test_stat(sample_1, sample_2, alpha))
    })

  return(sum(results<alpha)/num_samples) # fraction detected
}

# Create two slightly different populations (which happen to be Normal here).
population_1=rnorm(100000, mean=0, sd=1)
population_2=rnorm(100000, mean=0+0.5, sd=1)

expected_sample_size=20 # The expected size of the sample to be collected
boot_power=function(population_1, population_2, expected_sample_size,
                    some_test_statistic, alpha=0.05)
```

Figure 10.14 shows the results of a Monte Carlo simulation that tests for a difference in the mean of two samples of various sizes, each drawn from a different population (see `statistics/response-power.R` for the values calculated using an analytic solution, applicable for populations having a Normal distribution).

Obtaining good enough accuracy from a power analysis requires a good approximation of the likely characteristics of the sample obtained by an experiment. This information about the sample might be extracted from the results of related studies, a preliminary study or theory of the processes involved.

## 10.3 Describing a sample

A list of values can overwhelm readers with too much detail and techniques for compressing many values into a few, often just one value, are available.<sup>vii</sup> The few compressed values are known as *descriptive statistics* and the following are some common sample descriptions:

- a point estimate of a central value and its variability, e.g., mean and standard deviation,
- an equation fitted to the sample data according to some condition, e.g., minimising mean squared error,
- quartiles, a cluster of measurements based on where values are relative to other values in the sample, e.g., a box-and-whiskers plot such as Figure 8.20.

The mean and standard deviation are the two most commonly used descriptive statistics. It is incorrect to think that two distributions having the same mean and standard deviation will be very similar; see Figure 10.15.

### 10.3.1 A central location

Perhaps the most widely used, single value summary of a sample, derives from the idea of a *middle* or *central* location.

<sup>vii</sup> Plotting is a technique that often uses all the values and is the primary focus of Chapter 8.

- the mean, is perhaps the most commonly used central location; obtained by adding together the values, in a sample, and dividing by the number of values,
- the *median* is obtained by sorting the  $N$  values into numerical order and selecting the value of the  $\frac{N+1}{2}$ th element (if  $N$  is even the average of the middle two values is used),
- the *mode* is the value most likely to be sampled (R's mode function is unrelated to the statistical algorithm of that name, it returns the type or storage mode of an object). The modeest package contains functions for estimating various kinds of mode.

For symmetric distributions the values of the mean, median and mode are equal, while for asymmetric distributions, the three values can be very different.

When sample values are drawn from a unimodal distribution, the difference between the median and mean is less than or equal to  $\sqrt{0.6\sigma}$ , and for other non-unimodal distributions less than  $\sigma$ .

The difference between the median and mode is less than or equal to  $\sqrt{3}\sigma$ .

Unless the sample distribution is symmetric, it is not possible to sum multiple modes, e.g., cost estimates. For nonsymmetric distributions, adding underestimates the true value, e.g., for a Gamma distribution the mean is  $k\theta$  and the mode is  $(k-1)\theta$ , where  $k$  and  $\theta$  describe the Gamma distribution.

Figure 10.16 shows the distribution of execution times of the 1,000 input data sets from Chen et al.<sup>281</sup> If we are interested in an estimate of the execution time of a randomly chosen input data set, the median value, the point that equally divides the number of input data sets is the obvious choice. If we are interested in an estimate of the execution time most likely to be encountered, the value of the mode is the obvious choice.

Some distributions have such fat tails that the mean is infinite, e.g., the Cauchy distribution. In practice, the regularity with which very large values occur results in the mean value of a sample jumping around erratically, as new measurements are made. A distribution that does not have a finite mean may still have a median; the median is not affected by extreme values in the way the mean is, and any extreme values that do appear in a sample do not prevent the median converging to a fixed value.

The median absolute deviation is based around using the median as a robust estimation of variance; supported by the mad function.

The well-known algorithms for calculating the mean and standard deviation of a sample require that each value be independent of the others. When a sequence of values is serially correlated, i.e., the value of a measurement is related to the value of one or more immediately previous measurements, the calculated mean and standard deviation is biased. In the case of the mean, the uncertainty in its value grows for positive correlation and decreases for negative correlation. The upper plot in Figure 10.17 shows the fraction of this change for various sample sizes; it is based on an AR(1) model, where each value correlates with the immediately preceding value by an amount given in the legends on the right of the plot (see Section 11.10 for a discussion of AR models). A positive correlation causes the ratio of the sample standard deviation, relative to the population standard deviation, to be underestimated, while a negative correlation causes it to be overestimated (the lower plot in Figure 10.17 shows the fraction of this change).

The sandwich package supports the calculation of various error measures that are caused by serial correlation (e.g., the lrvar function calculates the error in the long term mean of a series).

**Compositional data:** The individual components of compositional data (i.e., data whose components always sum to a fixed value, such as percentages summing to 100%) are correlated and the mean of each component cannot be calculated independently of the other components. The mean function in the compositions package calculates the mean of compositional data.

Several methods of calculating the variance and standard deviation of compositional data have been proposed. The compositions package supports the mvar function, which calculates what is known as the *total variance* (or *generalized variance*) and the msd function which calculates the *metric standard deviation* (both return single values). The variation matrix includes information about the relationship between pairs of components and is returned by

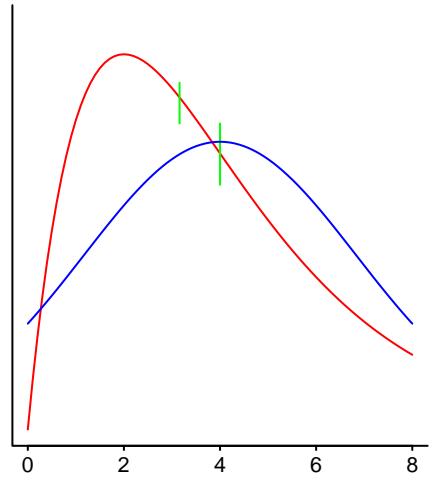


Figure 10.15: A Normal distribution with  $\text{mean}=4$  and  $\text{variance}=8$  and a Chi-squared distribution with four degrees of freedom having the same mean and variance (the vertical lines are at the distributions' median value). [code](#)

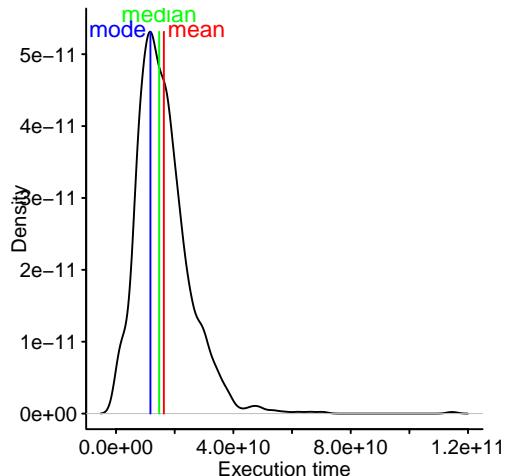


Figure 10.16: Density plot of execution time of 1,000 input data sets, with lines marking the mean, median and mode. Data kindly supplied by Chen.<sup>281</sup> [code](#)

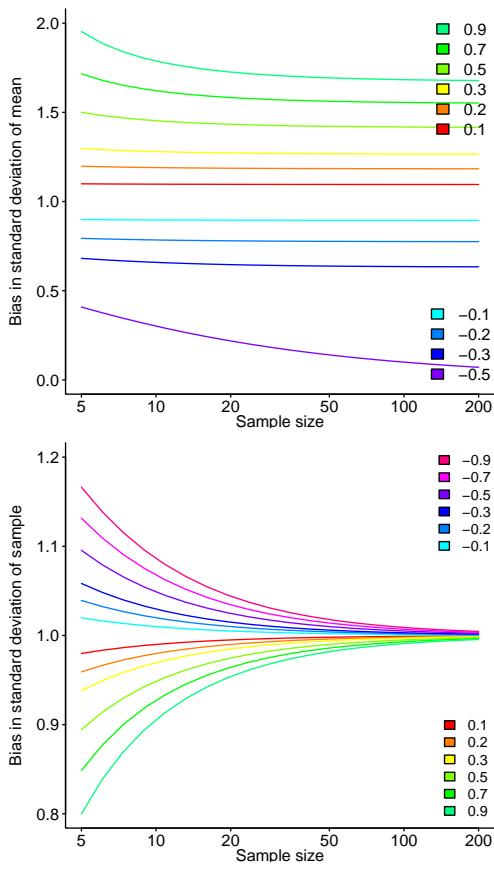


Figure 10.17: Impact of serial correlation, AR(1) in this example, on the calculated mean (upper) and standard deviation (lower) of a sample (the legends specify the amount of serial correlation). [code](#)

the variation function; see [statistics/composite-variation.R](#) for the variance calculation of the values plotted in Figure 5.32.

**Circular data:** Some measurements are made using a circular scale, with values that increase and wrap around from the maximum value to start again at the minimum value, e.g., angles take on values between 0 and 360.

The mean, if it exists, has a direction (or angle) and a length; Figure 11.77 illustrates a calculation of the mean of values drawn from a circular distribution.

### 10.3.2 Sensitivity of central location algorithms

Samples sometimes contain values that are noticeably different from the other values (e.g., much smaller or larger; which may or may not be the result of noise); the terms *outlier* or *influential observation* are used for such values. The percentage of sample values needed to cause a statistical estimator to produce an arbitrarily large (positive or negative) value is known as the *breakdown point*.

The breakdown point for the mean is proportional to  $\frac{1}{N}$ , i.e., no matter how many observations are made, it only takes one extreme value to produce a completely spurious result for the mean; the mean has the smallest breakdown point it is possible to have.

At the other end of the scale the median has a breakdown point of 0.5 (i.e., half of the measurements can have extreme value without affecting the value of the result) and for this reason the median is often recommended, over the mean, when measurements values are known to be very noisy. However, the median cannot be recommended for universal use because there are situations where it does not perform as well as the mean. For instance, when values are drawn from a discrete distribution whose mean is roughly half-way between measurable points, and the sample includes duplicate values, then most samples will have a median value slightly larger/smaller than the actual mean, i.e., the median is not evenly distributed across possible values in the way the mean is likely to be distributed; see Figure 10.18.

The probability of an outlier occurring depends on the reliability of the measurement process and the characteristics of the population being sampled. The following two techniques are robust in the presence of extreme values in a sample:

- *trimmed mean* removes a percentage of the largest and smallest values before calculating the mean of the remaining values (it has been found that 20% is a good value for general use). The `mean` function includes a `trim` argument for specifying the percentage to be trimmed,
- *winsorized mean* replaces rather than remove values. The values of the lowest X% are replaced with the lowest value that is just not within the specified percentage, and the values of the highest X% are replaced with the highest value just not within this percentage; the Winsorized mean is calculated using the updated list of values. The `psych` package contains functions that calculate various quantities using the Winsorized mean.

The trimmed and winsorized means may produce biased results when applied to samples drawn from a population having an asymmetric distribution.

### 10.3.3 Geometric mean

The *geometric mean* is defined as:

$$\text{Mean}_g = \left( \prod X \right)^{\frac{1}{N}}$$

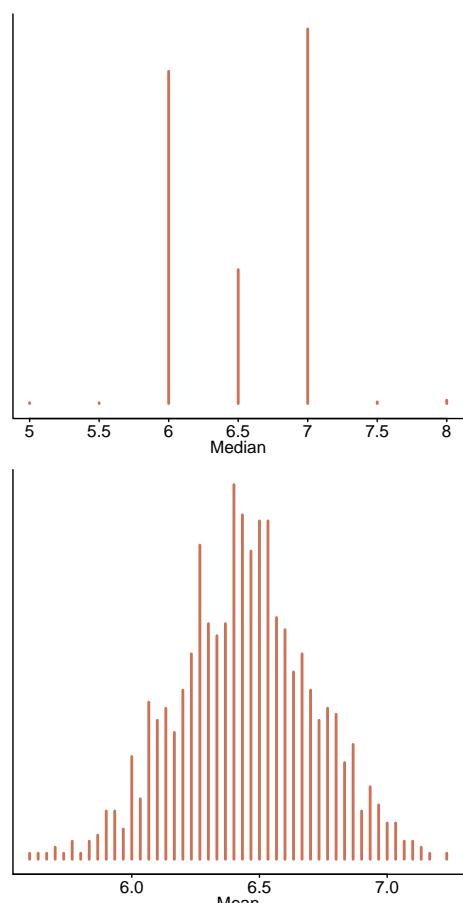
for instance, the geometric mean of 10, 100, 1000 is  $(10 \cdot 100 \cdot 1000)^{\frac{1}{3}} \rightarrow 100$ .

The geometric mean is preferred to the arithmetic mean when ranking ratios or normalised data (which is a kind of ratio), because it gives consistent results.

Consider the (invented) benchmark performance of the three systems in Table 10.2. Treating  $a$  as the base performance, what is the relative performance improvement of  $b$  and  $c$ ?

If the arithmetic mean is used, the performance ranking of  $b$  and  $c$ , relative to  $a$ , depends on whether the calculation used is a ratio of their means or the mean of their ratios. The

Figure 10.18: Number of sample median (upper) and mean (lower) values for 1,000 samples drawn from a binomial distribution. [code](#)



arithmetic means of the benchmark performance for each system is 50.5, 53.5 and 53: the ratio of these mean values is listed in the fourth column; the individual benchmark ratios for  $a$  and  $b$  are:  $\frac{2}{1}$  and  $\frac{105}{100}$ , and for  $a$  and  $c$ :  $\frac{3}{1}$  and  $\frac{103}{100}$ . The mean of these ratios is listed in the fifth column. Comparing columns four and five, shows that the ranking of  $b$  and  $c$  depends on the method of calculating the ratios. Also see Table 13.1.

system	integer	float	ratio of means	mean of ratios	geometric mean
a	1	100			10
b	2	105	$\frac{53.5}{50.5} \rightarrow 1.0594$	mean(2/1+105/100) -> 3.05	14.49
c	3	103	$\frac{53}{50.5} \rightarrow 1.0495$	mean(3/1+103/100) -> 4.03	17.58

Table 10.2: Inverted benchmark performance measurements of three systems and various methods of calculating relative performance. The relative performance of  $b$  and  $c$  depends on which mean is used.

If the geometric mean is used, the relative order of the final ratio is not order dependent.

Sometimes the arithmetic and geometric means produce the same benchmark rankings, e.g., a benchmark<sup>458</sup> of eight Intel IA32 processors used the arithmetic mean of ratios to compare results, the results from using the geometric means was not large enough to affect the relative ranking of processors for a given performance characteristic (see [benchmark/powervperfasp-los2011.R](#)).

The Geometric mean might be used when values cover several orders of magnitude, e.g., a geometric or logarithmic series (such as: 2, 4, 8, 16, 32, 64)

Methods for calculating the geometric mean include the expression `exp(mean(log(x)))` and the `geometric.mean` function in the `psych` package.

### 10.3.4 Harmonic mean

The *harmonic mean* is used to find the "average" of a set of ratios or proportions; it is defined as:

$$Mean_h = \frac{N}{\sum X^{-1}}$$

for instance, the harmonic mean of 1, 2, 3, 4, 5 is:  $\frac{5}{\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5}} \rightarrow 2.189781$

When there are two values the formula becomes:

$$Mean_h = 2 \frac{x \cdot y}{x + y}$$

which has the same form as the  $F_1$  score, or F-measure, used in information retrieval to combine the precision and recall:

$$F_1 = (1 + \beta^2) \frac{precision \cdot recall}{\beta^2 precision + recall}$$

Two methods for calculating the harmonic mean are: `1/mean(1/x)` and the `harmonic.mean` function in the `psych` package.

### 10.3.5 Contaminated distributions

That a sample, or the error present in the measurements it contains, is best described by a Normal distribution, is a common assumption that often goes unquestioned. Textbooks are filled with techniques that only exhibit the cited desirable attributes, when the Normality assumption holds. There is also the lure of analytic solutions to a problem, which again are often only possible when the Normality assumption holds.

Even when sample values appear to be drawn from a Normal distribution, a small percentage of contaminated values can have a dramatic effect on the value returned by a statistical algorithm.

The Contaminated Normal distribution is a mixture of values drawn from two Normal distributions, both having the same mean, but with 10% of the values drawn from a distribution

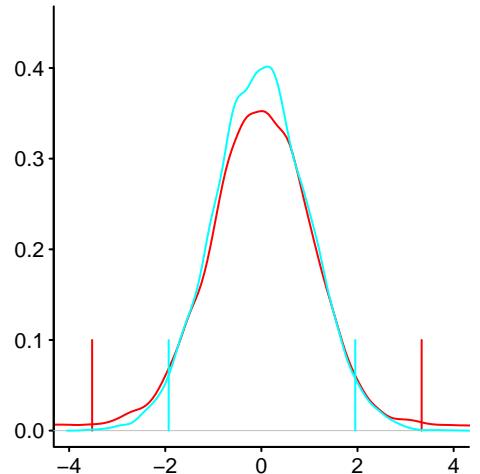


Figure 10.19: Density plot of two samples; samples either drawn from a Normal distribution or a Contaminated Normal distribution (i.e., values drawn from two normal distributions having a standard deviation five times greater than the other); the lines bounding the 95% quartile identify the color used for each plot. [code](#)

whose standard deviation is five times greater than the other. Figure 10.19 shows the kernel density of two samples, one containing 10,000 values drawn from a Normal distribution and the other containing 10,000 values from a Contaminated Normal distribution; visually they seem very similar (see `statistics/contam-norm.R` to learn the color used to plot each sample).

This Contaminated Normal distribution has a standard deviation that is more than three times greater than the Normal distribution from which 90% of the values are drawn. This illustrates that a Normal distribution contamination by just 10% of values from another distribution can appear to be Normal, but have very different descriptive statistics.

A number of tests are available for estimating whether sample values have been drawn from a Normal distribution. The Shapiro-Wilk test (the `shapiro.test` function), the Kolmogorov-Smirnov Test (the `ks.test` function)<sup>viii</sup> and the Anderson-Darling test are common encountered. A comparison of four normality tests<sup>1315</sup> found the Shapiro-Wilk test to be the most powerful normality test; see Figure 9.9.

When a data set contains very few values, even the Shapiro-Wilk test may fail to determine (e.g.,  $p\text{-value} < 0.05$ ) that sample values are not drawn from a Normal distribution. In the case of the Contaminated Normal distribution, samples containing only 10 values are considered to have a Normal distribution in around 30% of cases (based on  $p\text{-value} > 0.05$ ), with the percentage dropping to 10% for samples containing 20 values.

For an in depth analysis of the potential problems that outliers, skewed distributions and fat tails can cause see "Introduction to Robust Estimation & Hypothesis Testing" by Wilcox.<sup>1664</sup>

### 10.3.6 Meta-Analysis

Meta-analysis is the process of combining quantitative evidence from multiple studies to create more accurate estimates of the characteristics studied.

If descriptive statistics of each sample is the only information available, the mean and standard deviation can be pooled (creating a weighted single, combined value). The calculation is as follows (it assumes that each sample is independent of the other samples; at the time of writing, no built-in functions are available):

```
pooled_mean=function(df)
{
  return(sum(df$s_n*df$s_mean)/sum(df$s_mean))
}

pooled_sd=function(df)
{
  return(sqrt(sum(df$s_sd^2*(df$s_n-1))/sum(df$s_n-1)))
}

studies=data.frame(s_n=c(5, 10, 20),
                    s_mean=c(30, 31, 32),
                    s_sd=c(5, 4, 3))

pooled_mean(studies)
pooled_sd(studies)
```

Medical and social science experiments often measure one or more characteristics of a system before/after an event (e.g., a drug or social program). Various meta-analysis techniques have been created to deal with this kind of before/after study; the `meta` package contains support for this analysis. In software engineering, replicating studies of this kind is not (yet) a common occurrence.

A study by Sabherwal, Jeyaraj and Chowa<sup>1365</sup> performed a meta-analysis to compute a correlation matrix based on 612 findings from 121 studies published between 1980 and 2004...

File drawer problem...

---

<sup>viii</sup> Both included in R's base system.

## 10.4 Statistical error

The outputs from applying a statistical technique generally includes probabilities and it is the responsibility of the person doing the analysis to decide the cut-off probability below/above which an event is considered to have/have not occurred.

The two kinds of statistical error that can be made are:

- treating a hypothesis as true when it is actually false; the statistical term is making a *Type I* error, but *false positive* is more commonly used, and expressed in mathematics:  $P(\text{Type I error}) = P(\text{Reject } H_0 | H_0 \text{ true})$ ,
- treating a hypothesis as false when it is actually true; the statistical term is making a *Type II* error, but *false negative* is more commonly used, and expressed in mathematics:  $P(\text{Type II error}) = P(\text{Do not reject } H_0 | H_A \text{ true})$ , where  $H_A$  is an alternative hypothesis.

		Decision made	
		Reject $H$	Fail to reject $H$
Actual	$H$ true	Type I error	Correct
	$H$ false	Correct	Type II error

Table 10.3: The four states available in hypothesis testing and their outcomes.

The practical consequences of a statistical error depend on who is affected by the outcome of the decision made. For instance, consider the consequences of a manager's decision on whether to invest more time and money testing the reliability of a software system. An incorrect decision can result in more than losing the original investment (e.g., losing market share to a competitor); the bearer of any loss depends on the actual situation and the decision made, as Table 10.4 illustrates:

		Decision made	
		Finish testing	Do more testing
Actual	More testing needed	Customer loss	Ok
	Testing is sufficient	Ok	Vendor loss

Table 10.4: Finish/do more testing decision and outcome based on who incurs any loss.

### 10.4.1 Hypothesis testing

A hypothesis is an unverified explanation of why something is the way it is. Hypothesis testing is the process of collecting and evaluating evidence that may, or may not, be consistent with the hypothesis, i.e., positive and negative testing.<sup>ix</sup> Once enough evidence consistent with the hypothesis has been collected, people may feel confident enough to start referring to it as a theory or law.<sup>369</sup>

The most commonly used statistical hypothesis testing technique is based on what is known as the *null hypothesis*,<sup>x</sup> which works as follows:

- a hypothesis,  $H$ , having testable prediction(s) is stated,
- an experiment to test the prediction(s) is performed, producing data  $D$ ,
- assuming the hypothesis is true, the probability of obtaining the data produced by the experiment is calculated. The calculation made is:  $P(D|H)$ ; that is the probability of obtaining the data  $D$ , assuming that the hypothesis  $H$  is true.

If the calculated probability is less than or equal to some prespecified value, the hypothesis is rejected, otherwise it is said that *the null hypothesis has not been rejected* (i.e., the result of the experiment is not conclusive evidence that the null hypothesis is true).

<sup>ix</sup> Gigerenzer<sup>567</sup> discusses how people make decisions in an uncertain environment.

<sup>x</sup> As the market leader in hypothesis testing techniques, over many decades, this technique attracts regular criticism.<sup>310</sup> The criticism is invariably founded on widespread misuse of the null hypothesis ritual; misuse is the fate of all widely used techniques.

Expressed in code, the null hypothesis testing algorithm is as follows:

```
void null_hypothesis_test(void *result_data, float p_value)
{
    // H is set by reality, only accessed by running experiments
    if (probability_of_seeing_data_when_H_true(result_data) < p_value || !H)
        printf("Willing to assume that H is false\n");
    else
        printf("H might be true\n");
}

null_hypothesis_test(run_experiment(), 0.05);
```

A test statistic is said to be *statistically significant*, when it allows the null hypothesis to be rejected. The phrase "statistically significant" is often shortened to just "significant", a word whose common usage meaning is very different from its statistical one; this shortened usage is likely to be misconstrued when the audience is not aware that the statistical definition is being used, and treating the word as-if it is being used in its everyday meaning sense.

Statistical significance does not mean the pattern found by the analysis has any practical significance, i.e., the magnitude of the pattern detected may be so small as to make it useless for practical applications.

Running one experiment that produces a (statistically) surprisingly high/low p-value is a step in the process of increasing everybody's confidence that a hypothesis is true/false.

Replication of the results (i.e., repeating the experiment and obtaining similar measurements) provides evidence that the first experiment was not a chance effect; another boost in confidence. Replication by others, who independently set up and run an experiment, is the ideal replication (it reduces the possibility that unknown effects specific to a person or group influenced the outcome); an even larger boost in confidence.

There is a great deal of confusion surrounding how the results from a null hypothesis test should be interpreted. Studies have found<sup>569</sup> that people (incorrectly) think that one or more of the following statements apply:

- *Replication fallacy*: The level of significance measures the confidence that the results of an experiment would be repeatable under the conditions described. This is equivalent to saying:  $P(D|H) == 1 - P(D)$ , and would apply if the hypothesis was indeed true,
- the significance level represents the probability of the null hypothesis being true. This is equivalent to saying:  $P(D|H) == P(H|D)$ .

The Bayesian approach to hypothesis testing is growing in popularity and works as follows:

- two hypotheses,  $H_1$  and  $H_2$ , having testable prediction(s) are stated (the second hypothesis may just be that  $H_1$  is false),
- a non-zero probability is stated for the hypotheses being true,  $P(H_1)$  and  $P(H_2)$ , known as the *prior* probabilities,
- an experiment to test the prediction(s) is performed (producing data  $D$ ),
- the previously estimated probabilities, that  $H_1$  and  $H_2$  are true, is updated. The calculation uses Bayes theorem, which for  $H_1$  is:

$$P(H_1|D) = \frac{P(H_1)P(D|H_1)}{P(H_1)P(D|H_1) + P(H_2)P(D|H_2)}$$

The updated prior probability, on the basis of the experimental data, is known as the *posterior probability* of the hypothesis being true.

## 10.4.2 p-value

In a randomized experiment, the *p-value* is the probability that random variation alone produces a test statistic as extreme, or more extreme, than the one observed.

In a commercial environment, the choice of p-value should be treated as an input parameter to a risk assessment comparing the costs and benefits of all envisioned possibilities.

In many social sciences, the probability of the null hypothesis being rejected is required to be less than 0.05 (i.e., 5%, or slightly greater than  $2\sigma$ ),<sup>xii</sup> for a result to be considered worth publishing, while in civil engineering a paper describing a new building technique that created structures having a 1-in-20 chance of collapsing would not be considered acceptable. High energy physics requires a p-value below  $5\sigma \rightarrow 5.7 \cdot 10^{-7}$ , for the discovery of a new particle to be accepted.

As sample size increases, p-values will always become smaller. For instance, if some aspect of flipping a coin very slightly favours heads, given enough coin flips a sufficiently small p-value, for the hypothesis that the coin is not a fair one, will be obtained. There is no procedure for adjusting p-values for hypothesis testing using very large amounts of data.

When lots of measurement data, covering many variables, is available it is possible to go on a fishing expedition, looking for relationships between variables.<sup>1338</sup> The probability of finding one significant result, when comparing  $n$  pairs of variables, using a p-value of 0.05, is  $1 - (1 - 0.05)^n$  (which is 0.4, when  $n = 10$ ). When multiple comparisons are made, the base p-value needs to be adjusted to take account of the increased probability of noise being treated as a signal.

Perhaps the most common technique is the *Bonferroni correction*, which divides the base p-value by the number of tests performed. In the above example, the base p-value would be adjusted from 0.05 to  $\frac{0.05}{10} \rightarrow 0.005$ , to account for the possibility of each of the ten tests matching.

The `p.adjust` function supports p-value adjustment using a variety of different techniques.

Researchers know their work only has a chance of being accepted for publication, if the reported results have p-values below a journal's cut-off value. Given the use of published paper counts as a measure of academic performance, there is an incentive for researchers to run many slightly different experiments to find a combination that produces a sufficiently low p-value, that the work can be written up and submitted for publication<sup>801</sup> (a process known as *p-hacking*).<sup>xii</sup> One consequence of only publishing papers containing studies achieving a minimum p-value, is that many of the results are likely to be false (while a theoretical analysis suggests most are false,<sup>754</sup> an empirical analysis suggests around 14% of false positives for medical research<sup>768</sup>).

## 10.4.3 Confidence intervals

Many statistical techniques return a single number, a point value. What makes this number so special, would a value close to this number be almost as good an answer? If an extra measurement was added to the sample, how likely is it that the original number would dramatically change; what if one measurement were excluded from the sample, how much would that change the answer?

A *confidence interval* is an upper and lower bound on the numeric point value(s) returned by statistical technique. A common choice is the 95% confidence bound, the default value used by many R packages.

Numeric confidence intervals can be mapped into visual form by adding them to a plot. Figure 10.20 illustrates how confidence intervals provide an easier to digest insight into the uncertainty of a fitted regression model, compared to the single number that is the p-value. The red line shows a fitted regression model, whose predictor has a p-value of 0.02; the 95% confidence intervals in blue, showing how wide a range of lines could be said to fit the sample almost as well (i.e., any straight line bounded by the blue lines).

A confidence interval is a random variable, it depends on the sample drawn. If many 95% confidence intervals are obtained (one from each of many samples), the true fitted model is

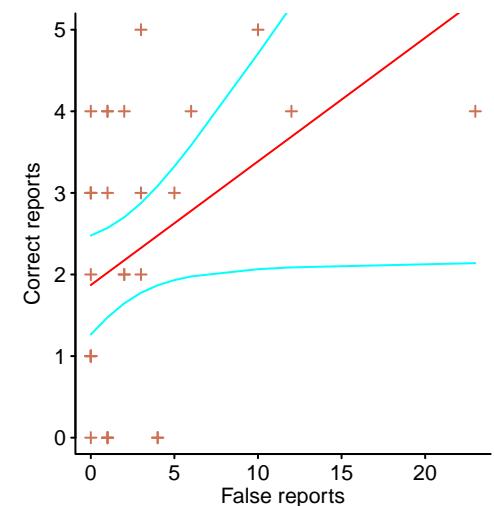


Figure 10.20: Regression model (red line; `pvalue=0.02`) fitted to the number of correct/false security code review reports made by 30 professionals; blue lines are 95% confidence intervals. Data from Edmundson et al.<sup>432</sup> code

<sup>xii</sup> Journals with high impact factors can be more choosy and some specify a p-value of 0.01.

<sup>xiii</sup> Which commercial company would not be willing to add warts to their software to keep an important customer happy?

expected to be included in this set of intervals 95% of the time (it is a common mistake to think that the confidence interval of one sample has this property). The probability that the next sample will be within the 95% confidence interval of the current sample, for a Normal distribution, is 84% or around 5 out of 6.<sup>344</sup>

A closed form formula for calculating confidence intervals is only known for a few cases, e.g., the mean of samples drawn from a Normal distribution; for a Binomial distribution a variety of different approximations have been proposed.<sup>1252</sup>

Built-in support for calculating confidence intervals, in R packages, is sporadic. Monte Carlo simulation can be used to calculate a confidence interval from the sample, e.g., the bootstrap. This approach has the advantage that it is not necessary to assume that sample values are drawn from any particular distribution. Figure 10.21 was created by fitting many models, via bootstrapping, and using color to indicate density of fitted regression lines.

#### 10.4.4 The bootstrap

The bootstrap is a general technique for answering questions about the uncertainty in the estimate of a statistic calculated from a sample (e.g., calculating a confidence interval or standard error).<sup>694</sup> Bootstrap techniques operate on a sample drawn from a population and cannot extract information about the population that is not contained in the sample, e.g., if the population contains reds and greens and a sample only contains reds, then the bootstrap will not provide any information about the greens.

The term bootstrapping is applied to the process by which a computer starts itself from an off-state. In statistics, it is used to describe a process where new samples are created from an existing sample; the term *resampling* is sometimes used.

The bootstrap procedure often starts by assuming there is no difference, in some characteristic, between samples; it then calculates the likelihood of two samples having the characteristic they are measured to have. The assumption of no difference requires that the items in both samples be *exchangeable*. Deciding which items, if any, in a sample are exchangeable is a crucial aspect of using the bootstrap to answer questions about samples.

Individual time series measurements contain serial correlations. The block bootstrap is one technique for applying bootstrap techniques to time series data. The `tsboot` function, in the `boot` package, supports the bootstrapping of time series data.

Estimating the confidence interval for the mean value of a sample is a good example of the basic bootstrap algorithm; the steps involved are as follows:

- create a sample by randomly drawing items from the original sample. Usually the items are selected with replacement (i.e., an item can be selected multiple times). When items are selected without replacement (i.e., can only be selected once), the term *jackknife* is used,
- calculate the mean value of the created sample,
- iterate the create/calculate cycle, say, 5,000 times,
- analyze the 5,000 mean values, to obtain the lowest and highest 2.5%. The 95% confidence interval for the mean of the original sample is calculated from this lowest/highest band (several algorithms, giving slightly different answers, are available).

The `boot` package supports common bootstrap operations, including the `boot.ci` function for obtaining a confidence interval from a bootstrap sample.

The distribution of the sample from which the bootstrap algorithm draws values is known as the *empirical distribution*.

The *bootstrap distribution* contains  $m^n$  possible samples, when sampling with replacement from  $m$  possible items to create samples containing  $n$  items; when the order of items does not matter, there are  $\binom{2m-1}{n}$  possible samples (a much smaller number).

The same bootstrap procedure can be applied to obtain confidence intervals on a wide range of metrics. Figure 10.22 shows confidence intervals for the kernel density plotted in Figure 8.14, and was produced by the `sm.density` function, in the `sm` package, using the following code:

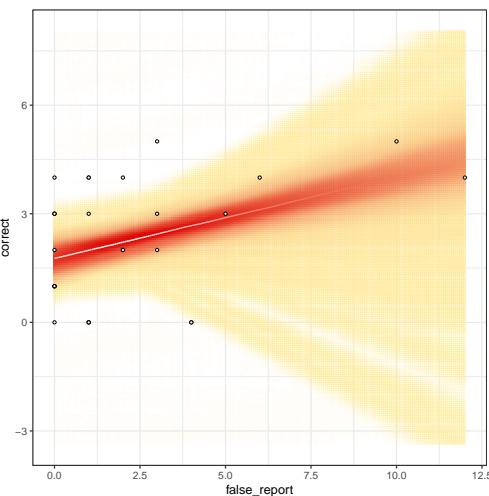


Figure 10.21: Bootstrapped regression lines fitted to random samples of the number of correct/false security code review reports made by 30 professionals. Data from Edmundson et al.<sup>432</sup> [code](#)

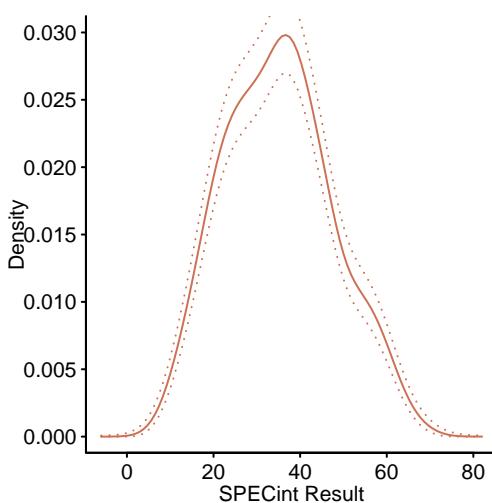


Figure 10.22: Kernel density plot, with 95% confidence interval, of the number of computers having the same SPECint result. Data from SPEC.<sup>1471</sup> [code](#)

```
library("sm")

res_sample=sample(cint$result, size=1000) # generate 1000 samples

sm.density(res_sample, h=4, col=point_col, display="se", rugplot=FALSE,
            ylim=c(0, 0.03),
            xlab="SPECint Result", ylab="Density\n")
```

### 10.4.5 Permutation tests

For small sample sizes, computers are fast enough for it to be practical to calculate a statistic (e.g., the mean) for all possible permutations of the items in a sample. This kind of test is known as a *permutation test*. Permutation tests do not have any preconditions on the distribution of the sample, other than it be representative of the population, and they return an exact answer.

Some techniques designed for manual implementation (e.g., Student's t-test) are approximations to the exact answer returned by a permutation test.

The coin package contains infrastructure for creating permutation tests and functions that perform common tasks (the names of these functions are derived from the names of the tests designed for manual implementation, e.g., spearman\_test and wilcox\_test).

The following permutation test calculates the likelihood that the professional experience of the two samples of subjects appearing in Figure 8.3 have different mean values:

```
library("coin")

# The default is alternative="two.sided",
# an option not currently listed in the Arguments section.
oneway_test(experience ~ as.factor(language), data=Perl_PHP, distribution="exact")
```

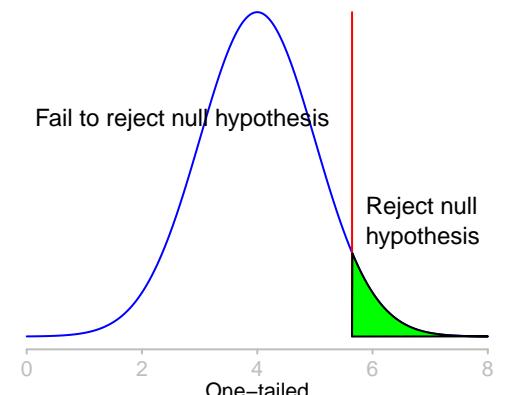
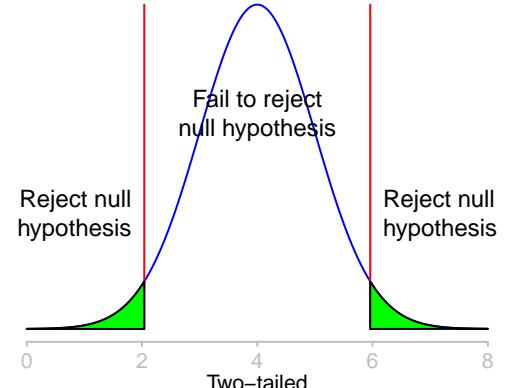
## 10.5 Comparing samples

The need to compare measurements, obtained from running experiments, kick started the development of statistics. The wide range of experimental designs (e.g., one/two/k samples, parametric/non-parametric and between/within subject), along with the need for practical manual solutions, resulted in the evolution of produced techniques designed to do a good job of handling each specific kind of comparison. This book assumes a computer is available to do the number crunching, and uses either regression (covered in Section 10.5.1) or the bootstrap.<sup>xiii</sup>

Samples may be compared to check whether they are the same/different, in some sense, or by specifically testing whether one sample is greater than, or less than, the other:

- in a *two-sided* test (also known as a *two-tailed* or *non-directional* test) the samples are checked for being the same or different, where an increase or decrease in some attribute is considered a difference. The percentage on each side, see Figure 10.23, is half the chosen p-value,
- in a *one-sided* test (also known as a *one-tailed* or *directional* test) the samples are checked for only one case, either an increase or a decrease in the measured attribute. The percentage on the one side, see Figure 10.23, is the chosen p-value,

A commonly encountered null hypothesis, when comparing two samples, is that there is no difference between them. In many practical situations a difference is expected, or hoped, to exist, otherwise no effort would have been invested in obtaining the data needed to perform the analysis.



<sup>xiii</sup> Other books tend to primarily cover the manual techniques: such as the t-test, which is a special case of multiple regression using an explanatory variable indicating group membership, and the Wilcoxon-Mann-Whitney test, which is essentially proportional odds ordinal logistic regression.

Experiments are often performed because a difference in one direction is of commercial interest. However, expecting or wanting a result that shows a difference in one direction is not sufficient justification for using a one-sided statistical test.

A one-sided test should only be used when the direction is already known, or when an effect in the non-predicted direction would be ignored. If an effect in a particular direction is expected, but an effect in the opposite direction would not be ignored (i.e., would be considered significant) a two-sided test should be used.

Some of the kinds of sample comparisons commonly made include:

- a level of confidence that sample values have been drawn from the same/different distribution,
- the difference,  $d_m$ , in the mean of two samples,
- the difference,  $d_v$ , in the variance of two samples,
- the correlation,  $C$ , between values, paired from two samples.

**Correlated measurements:** Many data analysis techniques assume that each measurement is independent of the other measurements in the sample.

An experiment that measures the same subject before and after the intervention (i.e., a within-subjects design; a between-subjects design involves comparing different subjects) involves correlated data. One technique for handling this kind of correlated data is mixed-effects models, discussed in Section 11.6.

Time dependent measurements may be correlated, with later measurements affected by earlier events that are not part of the benchmark (say). A correlation between successive measurements, where none should exist, either needs to be removed or taken into account during analysis. The Durban Watson test can be used to check for a correlation between successive measurements within each run. The `durbinWatsonTest` function, in the `car` package implements this test; see the discussion associated with Figure 11.22.

Time series analysis deals with sequentially correlated data, see Section 11.10.

### 10.5.1 Regression modeling

Using regression modeling to compare data may appear to be over-kill (it is used to analyse many of the datasets appearing in this book). When a computer is available to do the work, it makes sense to use the most powerful analysis techniques available; learning to apply the appropriate, less powerful, technique is a waste time.

A study by Potanin, Damitio and Noble<sup>1270</sup> refactored the Java Development Kit collection so that it no longer made use of incoming aliases (e.g., following the owner-as-dominator or owner-as-accessor encapsulation discipline). The DaCapo benchmark,<sup>170</sup> which contains 14 separate programs, was used to compare the performance of the original and refactored versions. The programs were each run 30 times, with measurements made during each of the last five iterations; this process is repeated five times, generating 25 measurements for each program for a total of 350 measurements.

The researchers claimed that their changes to the aliasing properties of the original code did not degrade performance. If the claim is true, the explanatory variable kind-of-refactoring, will have a trivial impact on the quality of the fitted regression model. The simplest model possible is based on the name of the program, and explains 99.9% of the variance (in this case the intercept is an unnecessary degree of freedom):

```
prog_mod=glm(performance ~ progname-1, data=dacapo_bench)
```

The fitted equation contains just the mean value of the runtime of each separate program, for all programs in the sample. The `summary` output is: `code`

```
Call:  
glm(formula = performance ~ progname - 1, data = dacapo)
```

Deviance Residuals:				
Min	1Q	Median	3Q	Max

```
-4174.6 -205.0 -9.0 116.6 3946.5

Coefficients:
Estimate Std. Error t value Pr(>|t|)
prognameavrora 22881.32 48.03 476.439 < 2e-16 ***
prognamebatik 2519.87 48.03 52.469 < 2e-16 ***
prognameeclipse 53660.53 48.03 1117.330 < 2e-16 ***
prognamefop 395.89 48.03 8.243 2.92e-16 ***
prognameh2 24100.39 48.03 501.823 < 2e-16 ***
prognamejython 15808.13 48.03 329.160 < 2e-16 ***
prognameluindex 708.00 48.03 14.742 < 2e-16 ***
prognamelusearch 7239.52 48.03 150.743 < 2e-16 ***
prognamepmd 4017.61 48.03 83.656 < 2e-16 ***
prognamesunflow 22788.81 48.03 474.513 < 2e-16 ***
prognametomcat 7672.11 48.03 159.750 < 2e-16 ***
prognametradebeans 27987.82 48.03 582.768 < 2e-16 ***
prognametradesoap 64888.58 48.03 1351.122 < 2e-16 ***
prognamexalan 26381.35 48.03 549.318 < 2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

(Dispersion parameter for gaussian family taken to be 345970)

```
Null deviance: 1.5873e+12 on 2100 degrees of freedom
Residual deviance: 7.2169e+08 on 2086 degrees of freedom
AIC: 32759
```

Number of Fisher Scoring iterations: 2

Adding kind-of-refactoring as an explanatory variable (see [regression/dacapo\\_progname.R](#) for details), finds that it is not significant on its own, but an interaction exists between it and a few programs (primarily sunflow). The model:

```
prog_refact_mod=glm(performance ~ progname+progname:refact_kind,
                     data=dacapo_bench)
```

explains 99.92% of the variance. There are 12 program/refactoring interactions with p-values less than 0.05 (out of 84 possible interactions), with most of these changing the estimated mean performance by around 1% and one making 8% difference (sunflow, see [[regression/-dacapo\\_progname\\_refact.R](#)]).

Building a regression model has enabled us to confirm that, apart from a few, small, interactions the various refactorings of the JDK did not change the DaCapo benchmark performance.

## 10.5.2 Comparing sample means

Comparing two samples, to check for a difference in their mean values, is perhaps the most common statistical test performed. The bootstrap is the general purpose tool to use to answer sample comparison questions; discussed in Section 10.4.4.

The nVidia GTX 970 is a popular graphics card, with many variations on the reference design being sold (during August 2016 there were 51 variants included in the 64,392 results for this card in the [UserBenchmark.com](#) database). Figure 10.24 shows the number of Reflection benchmark results reported for GTX 970 cards, from three third-party manufacturers.

The mean score of these Asus, MSI and Gigabyte cards are 176.2, 179 and 186.8 respectively. Are these differences more likely to be the result of random variation or by some real hardware/software difference?

The bootstrap can be used to answer this question, as follows.

Assume there is no difference in the mean performance of, say, MSI and Gigabyte on the Reflection benchmark. In this case the benchmark results (255 from MSI and 73 from Gigabyte) can be merged to form a sample of 328 results. Using this combined empirical sample perform the following:

- randomly select, with replacement, 328 items from the empirical sample,

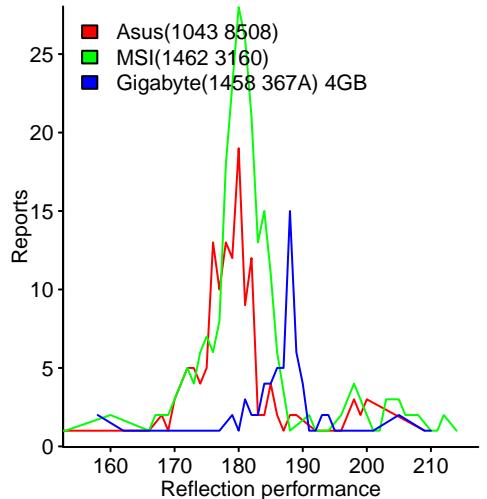


Figure 10.24: Number of Reflection benchmark results achieving a given score, reported for GTX 970 cards from three third-party manufacturers. Data extracted from [UserBenchmark.com](#). code

- divide this new sample into two subsamples, randomly selecting one to contain 255 items and the other 73 items,
- find the mean of the two subsamples, subtract the two mean values and record the result,
- repeat this process  $R$  times,
- count how many of these bootstrapped differences in the mean are greater than the differences in the means of the two cards; no assumption is made about the direction of the difference, i.e., this is a two-sided test.

The following code uses the `boot` function, from the `boot` package, to implement the above algorithm, with the user provided function (`mean_diff` in this case) that is called for each randomly generated sample (see [group-compare/UserBenchmark\\_compare.R](#)):

```
library("boot")

mean_diff=function(res, indices)
{
  t=res[indices]
  return(mean(t[1:num_MSI])-mean(t[(num_MSI+1):total_reps]))
}

MSI_refl=MSI_1462_3160$Reflection
Giga_refl=Gigabyte_1458_367A$Reflection

num_MSI=length(MSI_refl)      # Size of each sample
num_Giga=length(Giga_refl)
total_reps=num_MSI+num_Giga  # Total sample size

GTX_boot=boot(c(MSI_refl, Giga_refl), mean_diff, R = 4999) # bootstrap

refl_mean_diff=mean(MSI_refl)-mean(Giga_refl) # Difference in sample means
# Two-sided test
length(GTX_boot$t[abs(GTX_boot$t) >= abs(refl_mean_diff)]) # == E
```

The argument  $R$  specifies the number of resamples, with `boot` returning the result of calling `mean_diff` for each resample.

The likelihood of encountering a difference in mean values, as large as that seen in the MSI and Gigabyte performance (i.e., the p-value), is given by the equation:

$$\frac{E + 1}{R + 1}$$

where:  $E$  is the number of cases where the bootstrap sample had a larger mean difference. The result varies around:  $\frac{34+1}{4999+1} \rightarrow 0.007$  (the MSI/Asus comparison the value is:  $\frac{840+1}{4999+1} \rightarrow 0.17$ ).

If there were no difference in performance, a difference in mean value as large as that seen for MSI/Gigabyte is likely to occur 0.7% of the time. Based on a 5% cut-off, we can claim this percentage is so small that there is likely to be a real difference in performance. A mean difference at least as large as the MSI/Asus mean difference, is likely to occur 17% of the time, when there was no real difference in performance; a large enough percentage to infer that there is unlikely to be any difference in performance.

If a difference is thought likely to exist, the next question is the likely size of the difference and the confidence intervals on this value. A bootstrap procedure can be used to answer these questions.

Once two samples are considered to be different, items within each sample can only be treated as exchangeable with other items within the corresponding sample. The two subsample now have to be selected from their respective empirical samples, as in the following code (see [group-compare/UserBenchmark\\_mdiff.R](#)):

```
library("boot")

mean_diff=function(res, indices)
{
  t=res[indices, ]
  return(mean(t$refl[t$vendor == "Gigabyte"])- mean(t$refl[t$vendor == "MSI"]))
```

```

}

# Need to identify vendor used for each measurement.
MSI_refl=data.frame(vendor="MSI", refl=MSI_1462_3160$Reflection)
Giga_refl=data.frame(vendor="Gigabyte", refl=Gigabyte_1458_367A$Reflection)

MSI_Giga=rbind(MSI_refl, Giga_refl)

# Pass combined dataframe and specify identifying column
GTx_boot=boot(MSI_Giga, mean_diff, R = 4999, strata=MSI_Giga$vendor)

```

The `boot.ci` function calculates confidence intervals from the values returned by `boot` (in this case, the difference in mean values): [code](#)

```

> boot.ci(GTx_boot)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 4999 bootstrap replicates

CALL :
boot.ci(boot.out = GTx_boot)

Intervals :
Level      Normal          Basic
95%   ( 4.413, 11.206 )  ( 4.385, 11.154 )

Level      Percentile       BCa
95%   ( 4.475, 11.244 )  ( 4.875, 11.797 )
Calculations and Intervals on Original Scale
> mean(GTx_boot$t)
[1] 7.819614
> sd(GTx_boot$t)
[1] 1.732926

```

Deciding if, and when, items in a sample are exchangeable can be non-trivial and requires an understanding of the problem domain.

A study by Gandomani, Wei and Binhamid<sup>541</sup> investigated the accuracy of software cost estimates, made using both expert judgement and Planning Poker, on 15 projects in one company, and both expert judgement and Wideband Delphi in 17 projects in another company; Table 10.5 shows a subset.

Is there a difference in the estimates made using expert judgement and either of the other two techniques?

Project	Expert judgement	Planning Poker	Difference
P1	41	40	1
P4	60	56	4
P7	33	45	-12
P12	18	20	-2

Table 10.5: Effort estimates made using expert judgement and Planning Poker for several projects. Data from Gandomani et al.<sup>541</sup>

Each estimate is specific to one project, and it makes no sense to include estimates from other projects in the random selection process; estimates from different projects are not exchangeable. Possible ways of handing this include:

- treating each project as being exchangeable; the resampling could occur at the project level with both estimates for each selected project being used. This makes no sense, from the business perspective.
- randomly selecting from the set of estimates for each project. This possibility is not ruled out, from the business perspective, even though there are only two estimates for each project.

The following code randomly samples estimates for each project (see [group-compare/16.R](#)):

```

mean_diff=function()
{
  s_ind=rnorm(len_est_2) # random numbers centered on zero
  # Randomly assign estimates to each group
  expert=c(est_2$expert[s_ind < 0], est_2$planning.poker[s_ind >= 0])
  # Sampling with replacement, so two sets of random numbers needed
  s_ind=rnorm(len_est_2) # random numbers centered on zero
  poker=c(est_2$expert[s_ind < 0], est_2$planning.poker[s_ind >= 0])
  # The code for sampling without replacement
  # poker=c(est_2$expert[s_ind >= 0], est_2$planning.poker[s_ind < 0])
  return(mean(expert)-mean(poker))
}

est_mean_diff=abs(mean(est_2$expert)-mean(est_2$planning.poker))
len_est_2=nrow(est_2)

t=replicate(4999, mean_diff()) # Implement our own bootstrap

length(t[abs(t) > est_mean_diff]) # How many this extreme?

```

The p-value for a two-sided test between Expert and Planning Poker is 0.02 (see [group-compare/16.R](#)), which suggests there is a difference, but does not provide any information about the direction of difference.

A study by Jørgensen and Carelius<sup>800</sup> asked companies to bid on a software development project.<sup>xiv</sup> In the first round of bidding 17 companies were given a one-page description of user needs and asked to supply a non-binding bid; in the second round the original 17 companies plus an additional 18 companies (who had not participated in the first round) were given an 11-page specification (developed based on feedback from the first round) and asked to submit firm-price bids.

What difference, if any, did participating in the first round make to the second bids, submitted by the initial 17 companies (call them the A companies) and how did these bids compare to those submitted by the second sample of 18 companies bidding for the first time (call them the B companies)?

Figure 10.25 shows density plots of the submitted bids. The mean values were: kr183,051<sup>xv</sup> for initial bid from A companies, kr277,730 for final bid from A companies and kr166,131 for single bid from B companies.

Are the items in each sample (the companies asked to submit a bid) exchangeable? Small companies have lower operating costs than large companies; it is unrealistic to consider bids from small/large companies to be exchangeable. The size of companies involved in bidding were classified as small (five or fewer developers), medium (between 6 and 49 developers) and large (50 or more developers).

The call to boot has to include information on how the data is stratified (i.e., split into different levels). The argument strata is used to pass a vector of integer values specifying the strata membership of the values present in the first argument. Everything else stays the same, with boot treating members of each strata as exchangeable when generating new samples (see [group-compare/compare-bid.R](#)):

```

bid_boot=boot(comp_bid$Bid, mean_diff, R = 4999,
              strata=as.factor(comp_bid$CompSize))

```

The p-value, for the hypothesis that the mean values are the same, is:  $\frac{52+1}{4999+1} \rightarrow 0.01$ , i.e., a difference this large is surprising.

Jørgensen and Carelius proposed the hypothesis that the main factor controlling the size of the bids was the information contained in the project specification. I think this is rather idealistic, more practical considerations are discussed in Section 5.3.

Within subjects experiments using bootstrap... Given two samples,  $S_1$  where X was not performed and  $S_2$  where X was performed, the mean performance time of the samples, for instance, is likely to be different. How surprised should we be at this difference?

<sup>xiv</sup> Four of the companies that submitted a bid were selected to independently implement the project.

<sup>xv</sup> The exchange rate is approximately 10 Norwegian Krone to one Euro.

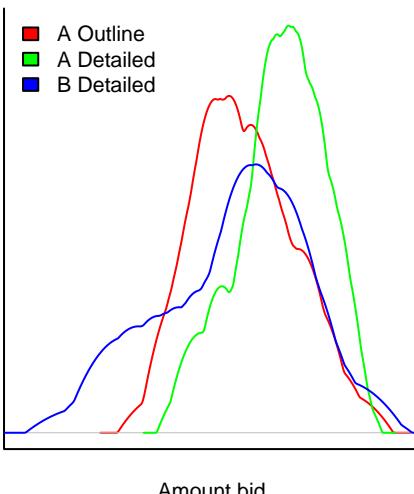


Figure 10.25: Density plots of project bids submitted by companies before/after seeing a requirements document. Data from Jørgensen et al.<sup>800</sup> [code](#)

Bootstrapping time series... using the `tsboot` rather than the `boot` function?

**Permutation tests:** When the two samples contain only a few items, it is practical to generate and test all possible item permutations.

A study by Grant and Sackman<sup>614</sup> measured the time taken for subjects to write a program using either an online or offline computer interface (this experiment was run during the 1960s mainframe era). Given 12 subjects, split into two groups of six, how likely is the difference in mean time between the online/offline use cases?

This question is about the population of people who took part in the experiment, not a wider population. For this population there are choose(12, 6) == 924 possible subject combinations. The following is an excerpt of an implementation of a two-sided test (see `group-compare/GS-perm-diff.R`):

```
subj_time=c(online$time, offline$time) # Combine samples
subj_mean_diff=mean(online$time)-mean(offline$time)

# Exact permutation test
subj_nums =seq(1:total_subj)
# Generate all possible subject combinations
subj_perms=combn(subj_nums, subj_online)

mean_diff = function(x)
{
  # Difference in mean of one combination of subjects
  mean(subj_time[x]) - mean(subj_time[!(subj_nums %in% x)])
}

# Indexing by column iterates through every permutation
perm_res=apply(subj_perms, 2, mean_diff)

# p-value of two-sided test
sum(abs(perm_res) >= abs(subj_mean_diff)) / length(perm_res)
```

For the Algebra program, 272 of the possible groups, of subject combinations, had a difference in mean time greater than, or equal, to that of the empirical sample. Because all possibilities have been calculated, the p-value is exact:  $\frac{272}{924} \rightarrow 0.2934$ .

The `coin` package provides this kind of exact calculation for many of the traditional group comparison tests, e.g., the `wilcoxonsign_test` function is the permutation test equivalent of the `wilcox.test` function (in the base library).

The bootstrap techniques used to answer questions about differences in the mean of two samples, can be generalised to a wide variety of comparison tests. A new comparison test can be implemented by replacing the `mean_diff` function used in the earlier examples (the requirement of exchangeability remains an integral requirement).

### 10.5.3 Comparing standard deviation

A study by Jørgensen and Moløkken<sup>804</sup> asked 19 professional developers to estimate the effort required to implement a task, along with an uncertainty estimate (i.e., minimum and maximum about the most likely value). Nine of the developers were explicitly instructed to compare the current task with similar projects they had worked on (they were also given a table that asked them to assess similarity within various percentage bands).

The visual appearance of the density plots, in Figure 10.26, suggests that there is a difference in the standard deviation of the estimates in the two samples. A bootstrap test, of the difference in the standard deviations of the two samples, can be implemented by replacing the `mean_diff` function used in the previous section, by the function `sd_diff` as follows; see `group-compare/simula_04sd.R`:

```
sd_diff=function(est, indices)
{
  t=est[indices]
  return(sd(t[1:num_A_est])-sd(t[(num_A_est+1):total_est]))
}
```

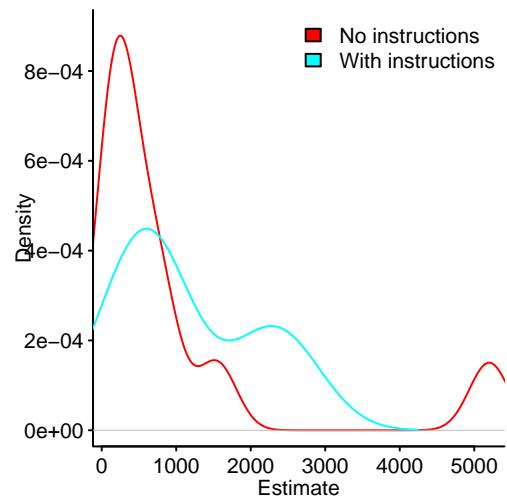


Figure 10.26: Density plot of task implementation estimates: with no instructions (red) and with instruction on what to do (blue). Data from Jørgensen et al.<sup>804</sup> code

The p-value, for the hypothesis that the standard deviations are the same, is:  $\frac{2170+1}{4999+1} \rightarrow 0.43$ , i.e., the difference is not that surprising.

The `ansari.test` function, in the `coin` package, <sup>xvi</sup> performs an *Ansari-Bradley Test* (a two-sample permutation test for a difference in variance); see [group-compare/simula\\_04\\_var.R](#).

### 10.5.4 Correlation

Correlation is a measure of linear association between variables, e.g., the extent to which one variable always increases/decreases when another variable increases/decreases. The range of correlation values is -1 (the variables change together, but in opposite directions) to 1 (the variables always change together), with zero denoting no correlation.

Correlation is related to regression, except that: it treats all variables equally (i.e., there are no response or explanatory variables), the correlation value is dimensionless and correlation is a linear relationship (i.e., there need not be any correlation between variables having a non-linear relationship, e.g., in the  $y = x^2$  relationship,  $y$  can be predicted  $x$ , but there is zero correlation between them).

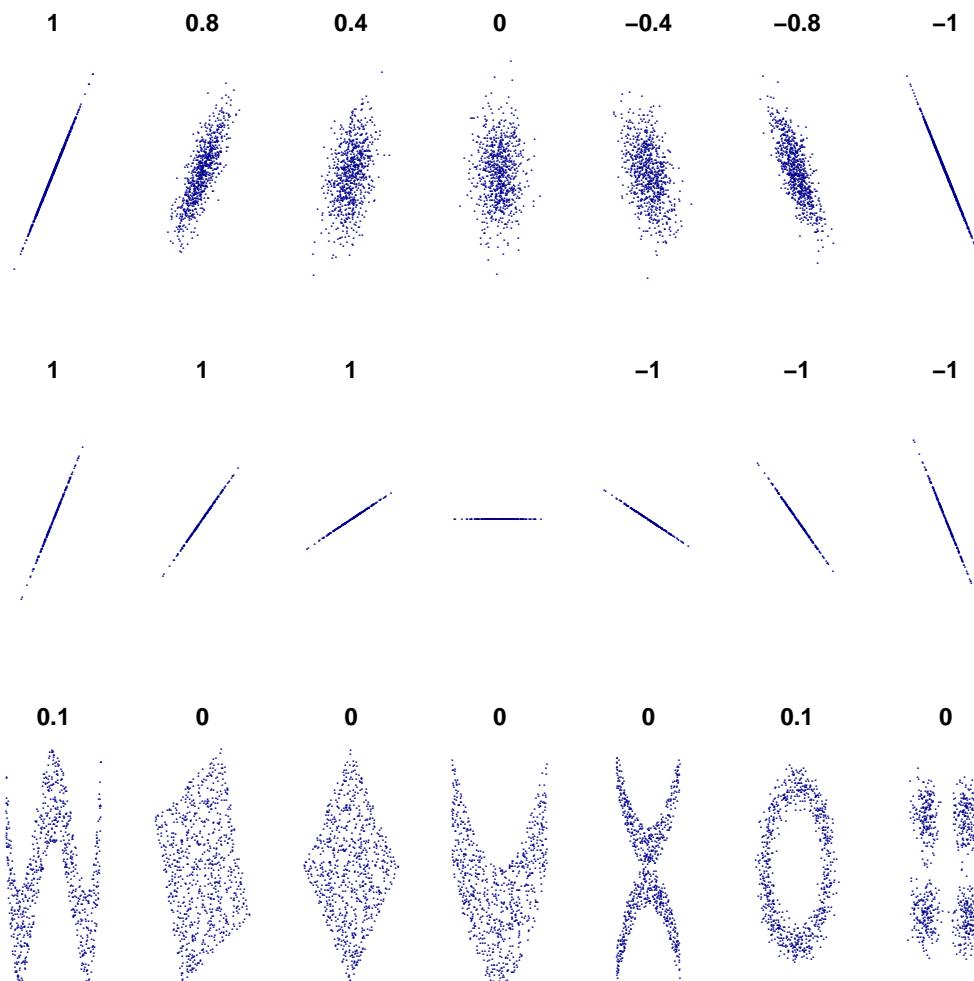


Figure 10.27: Examples of correlation between samples of two value pairs, plotted on x- and y-axis. [code](#)

Three commonly encountered correlation metrics are:

- *Pearson product-moment correlation coefficient*, (also known as Pearson's R or Pearson's r), which applies to continuous variables,
- Spearman's rho,  $\rho$  (a lowercase Greek letter), is identical to Pearson's coefficient except the correlation is calculated from the ranked values, i.e., the sorted order (which makes it immune to extreme values),

<sup>xvi</sup> The `ansari.test` function is included in R's base system.

- Kendall's tau,  $\tau$  (a lowercase Greek letter), is like Spearman's rho in that it is based on ranked values, but the calculation is based on the number of items sharing the same rank (i.e., relative difference in rank is not included in the calculation; Spearman's rho does include relative differences).

The `cor.test` function, included in the base system, supports all three coefficients and provides confidence interval.

**Dichotomous variables** When the result of a measurement has one of two values, the standard techniques for calculating correlation, which require that most if not all values be unique, cannot be used. It is possible to recast the problem in terms of probabilities, which means that the approach taken for every problem could be different.

The following is an example of one approach to a particular binary problem involving binary measurements.

One technique for having high reliability access files, is to host the files on two or more websites; if one site cannot be accessed, the file could be obtained from another site. The naive analysis suggests, that, if the average reliability of the websites is 95%, then the reliability of two paired sites would be 99.75%. However, this assumes the unavailability of each website is independent of its paired site.

A study by Bakkaloglu, Wylie, Wang and Ganger<sup>97</sup> had a client program read a file from over 120 websites every 10-minutes, between September 2001 and April 2002. They recorded whether the file was successfully accessed or not.

Most of the websites were available most of the time. Bakkaloglu et al proposed various techniques for calculating correlated failures, based on the probability that site  $X$  is unavailable when site  $Y$  is unavailable, i.e.,  $P(X\text{unavailable}|Y\text{unavailable})$ . The following example takes the mean value over all pairs of sites:

$$\begin{aligned} &= \text{mean}(P(X\text{unavailable}|Y\text{unavailable})) \\ &= \text{mean}\left(\frac{P(X\& Y\text{unavailable})}{P(Y\text{unavailable})}\right) \end{aligned}$$

The following calculates the average unavailability probability for one site paired with every other site; see [probability/reliability/web-avail.R](#):

```
given=web_down[ , ind]
others=web_down[ , -ind]

both_down=(others & given)

av_prob=mean(colSums(both_down)/sum(given))
```

Averaged over all pairs of sites the probability of one site being unavailable, when its pair is also unavailable, is 0.3 (at the 10-minute measurement point). Given that all accesses originated from the same client, it is not surprising that this probability is much higher than the average probability of one site being unavailable (0.1); all accesses start off going through the same internet infrastructure and problems in this infrastructure will affect access to all sites.

### 10.5.5 Contingency tables

Count data with categorical explanatory variables has a natural visual representation, as a table of numbers; these tables are known as *contingency tables*. In Table 10.6, each cell contains a count of the items in the sample having both the listed row and column attributes.

Use of contingency tables, as a means of reducing lots of data to a compressed visual form, is surprisingly common. Reasons for compressing lots of data to the form of a contingency table include: wanting to hide information (i.e., readers have to think about what is being presented), not knowing how to make the best use of the available information (i.e., the compressed form throws away potentially useful information). Analysis of the uncompressed data is likely to reveal more about it, than an analysis of the simplified form.

Sometimes the only available data is presented in a contingency table.

A study by Nightingale, Douceur and Orgovan<sup>1157</sup> investigated the characteristics of hardware failures over a very large number of consumer PCs. Table 10.6 shows a contingency table containing the available data, i.e., the number of system crashes believed to have been caused by hardware problems involving the system DRAM or CPU.

	DRAM failure	no DRAM failure
CPU failure	5	2,091
no CPU failure	250	971,191

Table 10.6: Number of system crashes of consumer PCs traced to CPU or DRAM failures. Data from Nightingale et al.<sup>1157</sup>

The traditional, manual friendly, technique for analyzing this kind of data is the chi-squared test ( $\chi$  is the Greek letter), which provides a yes/no answer.<sup>xvii</sup>

A regression model can be fitted to this data (even though there is not a lot of it), extracting more information than the chi-squared test. [code](#)

Call:

```
glm(formula = failures ~ CPU * DRAM, family = poisson, data = PC_crash)
```

Deviance Residuals:

```
[1] 0 0 0 0
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	13.786278	0.001015	13586.243	< 2e-16 ***
CPUTRUE	-6.140881	0.021892	-280.505	< 2e-16 ***
DRAMTRUE	-8.264818	0.063254	-130.661	< 2e-16 ***
CPUTRUE:DRAMTRUE	2.228858	0.452194	4.929	8.27e-07 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 2.6646e+06 on 3 degrees of freedom  
Residual deviance: -7.7825e-11 on 0 degrees of freedom  
AIC: 43.948

Number of Fisher Scoring iterations: 3

The information in the fitted model that is not immediately obvious from the numbers in the table, is that the crash rate is higher when both the CPU and DRAM fail (although, in this case, an obvious conclusion).

The regression approach makes it trivial to handle more rows and columns, as well as non-straight line fits. As the number of values in the table increases, it becomes more difficult to visually extract any patterns that may be present; a heatmap is one way of highlighting details, e.g., Figure 10.28.

There are a wide variety of techniques for comparing multiple contingency tables. Note: different pair comparison algorithms can give very different results.<sup>1535</sup>

## 10.5.6 ANOVA

Readers are likely to encounter the acronym ANOVA (*Analysis of variance*), an analysis technique which developed independently of linear regression and having its own specialized terminology. This technique was designed for manual implementation.

Functionally ANOVA and least squares are both special cases of the general linear model (ANOVA is a special case of multiple linear regression with orthogonal, categorical predictors; ANCOVA adds covariates to mix). A one-way analysis of variance can be thought of as a regression model having a single categorical predictor, that has at least two (usually more) categories.

<sup>xvii</sup> The `chisq.test` function is part of the base system; if your readership demands a chi-squared test, the `chisq_test` function in the `coin` package can be used to bootstrap confidence intervals.

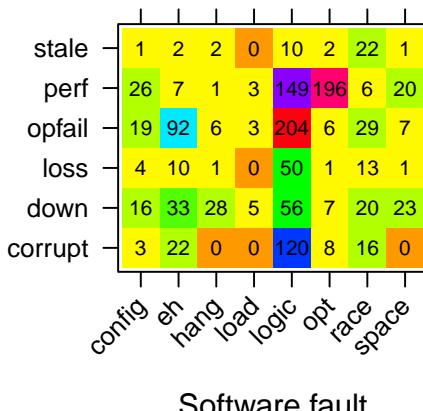


Figure 10.28: Number of software faults having a given consequence, based on an analysis of faults in Cassandra. Data from Gunawi et al.<sup>639</sup> [code](#)

Treating the various kinds of ANOVA models as special cases of the family of regression models, makes it possible to use the more flexible options available in regression modeling (e.g., easier handling of unequal group sizes, adjusting for covariates and methods for checking models).

The `anova` function generates ANOVA style output, when passed a model built using `glm` and some other regression model building functions. The `Anova` function in the `car` package...

One-way ANOVA focuses on testing for differences among a group of means; it evaluates the hypothesis that  $\alpha_i = 0$  in the following equation:

$$Y_i = \mu + \alpha_i + \varepsilon_i$$

where:  $\mu$  is the group mean,  $\alpha_i$  is the effect of the response variable on the  $i$ 'th group and  $\varepsilon_i$  is the corresponding error.



# Chapter 11

## Regression modeling

### 11.1 Introduction

Regression modeling is the default hammer used in this book to create the output from data analysis of software engineering data; Figure 11.1, gives a high level overview of the various kinds of hammers available in the regression modeling toolkit. Concentrating on a single, general technique, removes the need for developers to remember how to select from, and use, many special purpose techniques (which in many cases only return a subset of the information produced by regression modeling).

The arrow lines connect related regression techniques, based on the characteristics of the data they are designed to handle; the techniques highlighted in red are the common use cases for their respective data characteristics.

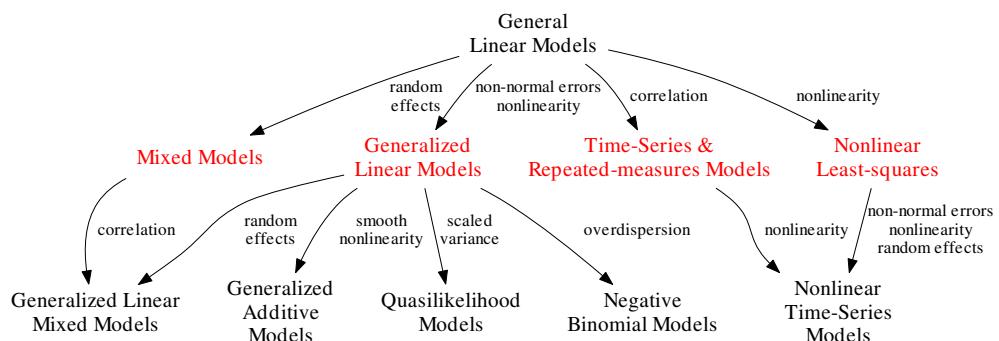


Figure 11.1: Relationship between data characteristics (edge labels) and applicable techniques (node labels) for building regression models.

Regression modeling is powerful enough to fit almost any data to within any selected error bounds, which means overfitting is an ever present danger; model validation (e.g., how well a model might fit new data or an estimation of the benefit obtained from including each coefficient in a model) is an important self-correcting step.

As always, it is necessary to remember the adage: “All models are wrong, but some are useful.”

The main reasons for building a regression model are:

- understanding: structuring the explanatory variable(s) in an equation that can be used to interpret the impact of they have on the response variable, i.e., build an understanding of the processes that influence the response variable to behaves the way it does,
- prediction: that is predicting the value of the response variable, for values of the explanatory variables that have not been measured.

The focus of interpretive modeling is understanding why, which creates a willingness to trade-off prediction accuracy for model simplicity, while the focus of predictive modeling is accuracy of prediction, which creates a willingness to trade-off understanding of behavior for greater accuracy.

Understanding is the primary focus for the model building in this book; builders of computing systems are generally interested in controlling what is happening and control requires understanding; predicting is a fall back position. Model building for prediction is often easier than building for understanding, once readers master building for understanding they will not find it difficult to switch to a predictive focus.

Regression models contain a *response variable*, one or more *explanatory variables*<sup>i</sup> and some form of error term.

The *response variable* is modeled as some combination of *explanatory variables* and an additive or multiplicative error term (the error term associated with each explanatory variable represents behavior not accounted for by the explanatory variable; different kinds of regression model make different assumptions about the characteristics of the error).

It is always possible to concoct a model that fits some data to within any error tolerance, i.e., the amount of variation in the measurements used, that the model does not explain. It is very important to always ask how well a model is likely to fit all the data likely to be encountered, not just the data used to build it.

If a sample contains many variables, then it is sometimes possible to build a model only using a few of these variables, that has an impressive fit to the chosen response variable. A study by Zeller, Zimmermann and Bird<sup>1698</sup> built a fault prediction model whose performance was comparable to the best available at the time. The model used four explanatory variables to predict the probability of a fault being reported in the source code contained in each file; the explanatory variables were the percentage occurrence of each of the characters IROP in each file. The model was *discovered* by checking how good a job every possible character did at predicting fault probability, and picking those that gave the best fit.

## 11.2 Linear regression

The simplest form of regression model is linear regression, where the *response variable* is modeled as a linear combination of *explanatory variables* and an additive error (the error terms are assumed to be independent and identically distributed;  $\varepsilon$  denotes the total error). The equation is:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon$$

Note that the term *linear* refers to the coefficients of the model, i.e.,  $\beta$ , not the form taken by the explanatory variables which may have a non-linear form, as in:

$$y = \alpha + \beta x^2 + \varepsilon$$

or:

$$y = \alpha + \beta \log(x) + \varepsilon$$

A linear model is perhaps the most commonly used regression model, reasons for this include:

- many real world problems exhibit linear behavior, or a good enough approximation to it for practical purposes, over their input range,
- they are much easier to fit manually than more sophisticated models and until recently software to build other kinds of models was not widely available,
- they can generally be built with minimal input from the user (apart from having to decide which column of the data to use as the response variable).

The `glm` function<sup>ii</sup> builds a linear model and the use common case requires two argument value, a formula expressing a relationship between variables (response variable on the left and explanatory variable(s) on the right) and an object containing the data (this object is

---

<sup>i</sup> Books that focus on the predictive aspect of models use the term *prediction variable* or just *predictor*, while those that focus on running experiments use terms such as *control variables* or just the *controls*.

<sup>ii</sup> Many books start by discussing the `lm` function, rather than `glm`, because the mathematics that underpins it is easier to learn, another reason for this is herd mentality, it's what everybody else does; if you dear reader want to learn this mathematics I recommend taking this approach. As its name implies the Generalised Linear Method has a wider range of applicability and its use here is in line with the aim of teaching one technique that can be used everywhere. Also, the mathematics behind `glm` makes fewer assumptions about the sample characteristics, e.g., it does not have the precondition that the variance in the error to be constant (which `lm` does).

required to contain columns whose names match the identifiers appearing in the formula). The formula has the form of an equation, with the  $=$  symbol replaced by  $\sim$  (pronounced *is distributed according to*) and the coefficients  $\alpha$  and  $\beta$  are implicitly present, i.e., they do not need to be explicitly specified in the code.

The following code uses `glm` to build a model showing the relationship between the number of lines of source code (*sloc*) in FreeBSD and the number of days elapsed since the project started (in 1993):

```
BSD_mod=glm(sloc ~ Number_days, data=bsd_info)
```

The fitted equation is:

$$E[sloc] = \alpha + \beta \times Number\_days$$

where  $E[sloc]$  is the expected value of *sloc* (the error term is discussed below).

Figure 11.2 shows the measured data points and a straight line based in the coefficients contained in the object returned by `glm`.

The `summary` function takes the object returned by `glm` and prints details about the fitted model;<sup>iii</sup> the following is for the model fitted to the FreeBSD data: [code](#)

Call:

```
glm(formula = sloc ~ Number_days, data = kind_bsd)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-82990	-32136	-3609	35389	87324

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	1.139e+05	1.171e+03	97.24	<2e-16 ***
Number_days	3.937e+02	4.205e-01	936.33	<2e-16 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 1657283104)

Null deviance: 1.4610e+15 on 4826 degrees of freedom  
 Residual deviance: 7.9964e+12 on 4825 degrees of freedom  
 AIC: 116172

Number of Fisher Scoring iterations: 2

The table following the **Coefficients:** header, in the `summary` output, lists the fitted values for  $\alpha$  and  $\beta$  (Intercept and *Number\_days* respectively), the standard error in these estimates (Std. Error) and the probability that, if the true value of the coefficient was zero, the estimated value would have occurred by chance (in the  $Pr(>|t|)$  column).

The values listed in the `summary` output can be plugged into the model formula to give the following fitted equation:

$$sloc = 1.139 \cdot 10^5 + 3.937 \cdot 10^2 Number\_days$$

The fit between the model and the data is not perfect and the following are the two forms of uncertainty, or variation, present in the model:

1. Uncertainty in the values of the model coefficients. The values listed in the Std. Error column denote one standard deviation, which when added to the model gives the following:

$$sloc = (1.139 \cdot 10^5 \pm 1.171 \cdot 10^3) + (3.937 \cdot 10^2 \pm 4.205 \cdot 10^{-1}) Number\_days$$

2. Uncertainty caused by the inability of the explanatory variable used in the model to explain everything. This uncertainty is the  $\varepsilon$  appearing in equation <??>; the term *residual* is used to denote this quantity. In the general case it is unlikely that  $\varepsilon$  will have a fixed value over the range of values supported by a model and `glm` does not return any value(s).

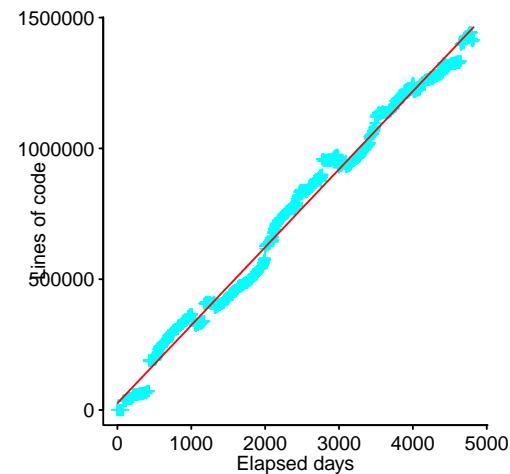


Figure 11.2: Total lines of source code in FreeBSD by days elapsed since the project started (in 1993). Data from Herraiz.<sup>[1529](#)</sup> [code](#)

<sup>iii</sup> Only a few digits of the estimated values are printed by default.

In Figure 11.2 the variations in the unexplained error,  $\epsilon$ , appear to be small. The `aov` function can be used to obtain a single fixed value; it returns 40,710 as the residual standard error. The equation, including this estimate of the residual is:

$$sloc = 1.139 \cdot 10^5 + 3.937 \cdot 10^2 Number\_days \pm 4.071 \cdot 10^4$$

In other words, the difference between measured values and values calculated using this fitted model are predicted to have a standard error of  $4.071 \cdot 10^4$ .

The object returned by the call to `glm` can be used to make predictions and these can be overlaid on the output from an earlier call to `plot`, as follows:

```
BSD_pred=predict(BSD_mod) # uses fitted model and measured values
lines(BSD_pred, col="red") # x-axis starts at 1 and increment
```

The `predict/lines` approach follows this book's aim of using techniques that work for the general case. Plotting a fitted straight line is such a common operation that there is a function for doing just that, e.g., `abline(reg=BSD_mod, col="red")`, but this does not always work when the axis have been scaled in some way and is of no use for fitted models that are more complicated than a straight line.

Before being carried away with the high degree of agreement between this model and the data, it is important to remember that the model has a number of characteristics that do not reflect reality, including:

- source code does not spontaneously grow of its own accord and the only justification for treating *number of days* as an explanatory variable is that the resulting model provides potentially interesting insight into the rate of growth of these software systems.
- when it started the BSD project contained zero lines of code, but this model has an Intercept of  $1.39 \cdot 10^5$ ,
- the model shows the number of lines increasing for ever, at a constant rate, whereas at some point in the future growth must slow down and eventually stop,
- it says nothing about large amounts of code being added/removed over very short periods (known to exist because of the visible breaks in the connectedness of the plotted values).

While the model has various disconnects with reality, it does provide strong evidence that growth has been remarkable constant over a long period. Unless there are seismic changes within the FreeBSD development world, the constant rate of code growth would be expected to continue to hold for a non-trivial number of days into the future.

Fitting a model to the data marks the start of the next stage of the analysis; creating viable explanations for the processes that could have produced the behavior found. Some of the factors and processes that might be involved in driving FreeBSD's essentially constant rate of growth over 20 years include:

- developers working on the system have continually discovered new functionality to add,
  - if there has always been functionality to add, why haven't more developers become involved, increasing the rate of growth until there is less to do?
  - to what extent is the continual stream of new hardware devices responsible for driving growth?
- what are the bottlenecks that have prevented increases in growth rate, when the resources have been available?
  - has growth rate remained constant because the developers working on the systems have remained constant?
  - is there a buffer of code waiting to be released, whose growing and shrinking hides an internal growth rate that is much more variable than the externally visible rate?

The questions answered by the analysis of one set of measurements invariably raises more questions, whose answers require more data.

The call to `summary`, passing the value returned by `glm`, is an example of function overloading in action. The value returned by `glm` has class `glm`, which, when passed as an argument to `summary`, results in `summary.glm` being called; a call to `predict` results in `predict.glm` being called (function overloading is the most common use of object oriented constructs in R programs; the use of a period in the function name is a naming convention followed by the implementers and not something that changes the behavior of the R compiler).

Some readers of data analysis may find a visual presentation of the coefficients of a fitted model, along with their standard error, easier to process. The `sjPlot` package offers a variety of options for plotting of fitted model information.

### 11.2.1 Scattered measurement values

In the FreeBSD analysis, the measurements ran together in a way that created a visually recognizable line. The common case is not always so accommodating, and often when many samples are plotted a scattering of visually disjoint points appears; viewed as a whole a general trend may emerge.

A study by Kampstra and Verhoeft<sup>819</sup> investigated the estimated cost and duration of 73 large Dutch federal IT projects.<sup>iv</sup> Figure 11.3 shows that few of the measurement points are close to the (red) fitted line returned by `glm`; the variability of the measured values is much larger than that for the FreeBSD data. While numeric estimates of the uncertainty present in the fitted model are readily available, interpreting these numeric values requires a degree of effort and some experience. A confidence interval provides an easy to interpret visual representation of the uncertainty in a fitted model.

The kind of uncertainty, in the fitted model, of interest will depend on whether the model is built to gain understanding or make predictions:

- when understanding is the priority, the confidence interval of interest involves the estimated model coefficients:

- a call to `predict` with the argument `se.fit=TRUE`, returns the standard error for each fitted value. Multiplying `se.fit` by `qnorm`,<sup>v</sup> converts the returned value to a 95% confidence interval (in this case, 2.5% above and below the fit; the two `qnorm` values differ only in sign because the Normal distribution is symmetrical), i.e., there is a 95% expectation that the actual model fits within the interval enclosed by these lower/upper bounds. `qnorm(0.975)==1.96` and the literal value is often used (sometimes the value 2 is treated as a sufficiently close approximation).<sup>vi</sup>

```
fed_pred=predict(fed_mod, newdata=list(log.IT=1:7, log.IT_sqr=(1:7)^2),
                 se.fit=TRUE)
lines(fed_pred$fit, col="green")      # fitted line
# CI above and below
lines(fed_pred$fit+qnorm(0.975)*fed_pred$se.fit, col="green")
lines(fed_pred$fit+qnorm(0.025)*fed_pred$se.fit, col="green")
```

- the `confint` function in the `MASS` package or the `boot.ci` function in the `boot` package, can be used to obtain a point estimate of the confidence interval of the fitted model coefficients.

- when prediction is the priority, the interval is known as the *prediction interval*; the bounds between which newly measured values are expected to appear. Two sources of uncertainty are added to calculate the prediction interval: uncertainty in the model coefficients (i.e., the confidence interval) plus the variance in the data not explained by the fitted model; the calculation is (the `predict` function can perform this calculation for a few types of fitted models):

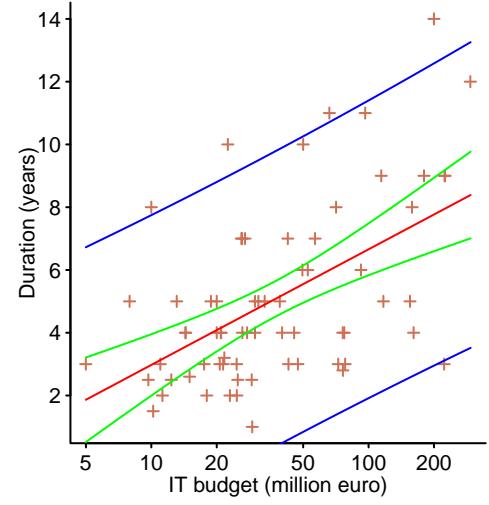


Figure 11.3: Estimated cost and duration of 73 large Dutch federal IT projects, along with fitted model and 95% confidence intervals (green for the bounds of the fitted line and blue for the bounds of any new measurements). Data from Kampstra et al.<sup>819</sup> code

<sup>iv</sup> They discovered there was a lot of uncertainty in the estimates given.

<sup>v</sup> This calculation assumes that the measurement error has a Normal distribution, the default assumption made by `glm` when building a model.

<sup>vi</sup> For small sample sizes a call to `qt` may be more accurate.

```
# print.aov also calculates it from residuals returned by glm...
MSE=sum(fed_mod$residuals^2)/(length(fed_mod$residuals)-2)
# Variances, but not sd, can be added
pred_se=sqrt(fed_pred$se.fit^2+MSE)
lines(fed_pred$fit+1.96*pred_se, col="blue")
lines(fed_pred$fit-1.96*pred_se, col="blue")
```

When measurement values and an associated fitted regression line are plotted, it is easy to visually fixate on the line and forget about the associated uncertainties. Including a confidence band as part of a plot provides a vivid visual reminder of the uncertainty in the fit.

Plotting values does not always reveal an obvious pattern in the distribution of points. The absence of a visual pattern may be because no relationship exists between the response and explanatory variables, or because the noise in the data is much greater than the signal (i.e., a relationship that exists is swamped by noise present in the measurements).

How much random scattering of measurement values has to exist before a fitted regression model can be said to be not worth bothering about?

The `glm` function, and many other model building functions available in R, is capable of fitting models to data points that are randomly distributed. For instance, Figure 11.4 shows the number of updates and fixes made in various Linux versions released between early 2011 and 2012. The standard error of the fitted line shows that its slope could have a positive or negative value.

The output from `summary` shows how poor the fit actually is; the `Pr(>|t|)` column lists the p-value for the hypothesis that the coefficient in the corresponding row is zero, i.e., that no relationship was found to exist for that component of the model. [code](#)

Call:

```
glm(formula = Fixes ~ Total.Updates, data = cleaned)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-310.60	-223.67	0.48	184.51	525.26

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	356.233	101.522	3.509	0.0016 **
Total.Updates	-4.464	8.478	-0.526	0.6029

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 60685.71)

```
Null deviance: 1655335 on 28 degrees of freedom
Residual deviance: 1638514 on 27 degrees of freedom
AIC: 405.62
```

Number of Fisher Scoring iterations: 2

## 11.2.2 Discrete measurement values

Regression models are not limited to fitting continuous numeric explanatory variables, variables having nominal values (i.e., discrete) can also be included in a fitted model.

A study by Cook and Zilles<sup>324</sup> investigated the impact of compiler optimization flags on the ability of software to continue to operate correctly, when subject to random bit-flips, i.e., simulating random hardware errors; 100 evenly distributed points in the program were chosen and 100 instructions from each of those points were used as fault injection points, giving a total of 10,000 individual tests run, for each of 12 programs from the SPEC2000 integer benchmark compiled using gcc version 4.0.2 (using optimization options: 00, 02 and 03) and the DEC C compiler (called *osf*).

The fitted model has percentage of correct benchmark program execution as the response variable<sup>vii</sup> and optimization level as the explanatory variable; the call to `glm` is unchanged:

```
bitflip_mod=glm(pass.masked ~ opt_level, data=bitflip)
```

The summary output of the fitted model is: `code`

Call:

```
glm(formula = pass.masked ~ opt_level, data = bitflip)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-12.6689	-2.8454	-0.3478	4.4017	8.1100

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	28.589	1.825	15.665	< 2e-16 ***
opt_level02	9.161	2.581	3.550	0.00112 **
opt_level03	7.429	2.581	2.878	0.00677 **
opt_level0sf	11.642	2.414	4.822	2.74e-05 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 29.97578)

Null deviance: 1785.8 on 38 degrees of freedom

Residual deviance: 1049.2 on 35 degrees of freedom

AIC: 249.07

Number of Fisher Scoring iterations: 2

Plugging the model coefficients into the regression equation we get:

`pass.masked = 28.6 + 9.2 × DO2 + 7.4 × DO3 + 11.6 × Dosf`

where:  $D_i$ , known as a *dummy variable* or *indicator variable*, take one of two values:

$$D_i = \begin{cases} 1 & \text{optimization flag used} \\ 0 & \text{optimization flag not used} \end{cases}$$

The value for optimization 00 is implicit in the equation, it occurs when all other optimizations are not specified, i.e., its value is that of the intercept.

The standard error in the O2 and O3 compiler options is sufficiently large for their respective confidence bounds to have significant overlap; suggesting that these two options have a similar impact on the behavior of the response variable.

### 11.2.3 Uncertainty only exists in the response variable

Many of the algorithms used to fit regression models attempt to minimise the difference between the measured points and a specified equation. For instance, least-squares minimises the sum of the squares of the distance along one axis between each data point and the fitted equation;<sup>viii</sup> alternative minimization criteria are discussed later, e.g., giving greater weight to positive error than negative error.

An important, and often overlooked, detail, is that many regression techniques assume that the values of the explanatory variable(s) contain no uncertainty (i.e., measurements are exact), with all uncertainty,  $\varepsilon$ , occurring in the response variable (see the [first equation](#) at the start of this chapter).

A consequence of assuming uncertainty only exists in the response variable, is that the equation produced by fitting a model that specifies, say,  $X$  as the explanatory variable and  $Y$  the response variable will not be algebraically consistent with a model that assumes  $Y$  is the explanatory variable and  $X$  the response variable. That is, algebraically transforming the first equation produces an equation whose coefficients are different from the second.

<sup>vii</sup> Percentage correct is always between 0 and 100%; technically correct techniques for handling response variables having a lower and upper bound are discussed in Section 11.3.6.

<sup>viii</sup> Minimising the sum of squares in the error has historically been popular because it is a case that can be analysed analytically.

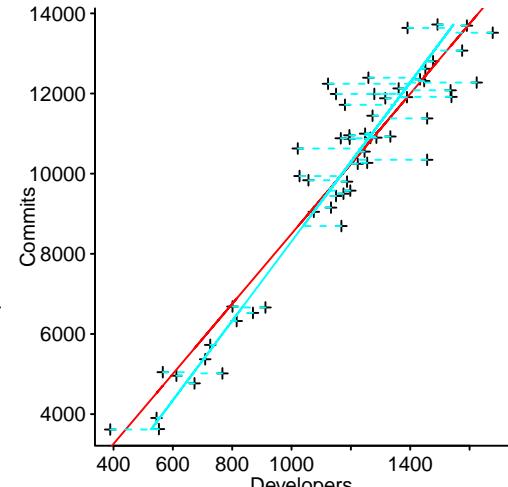
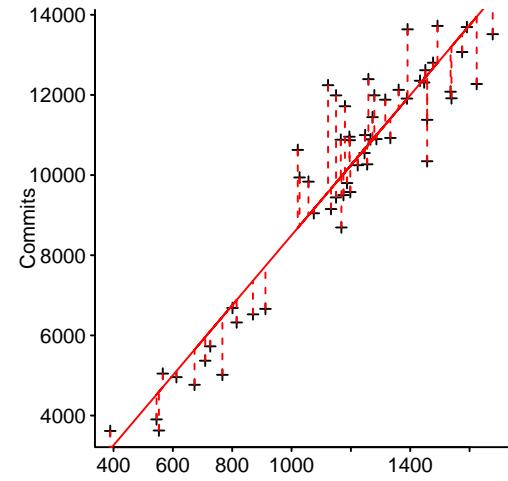


Figure 11.5: The number of commits made and the number of contributing developers for Linux versions 2.6.0 to 3.12. The green line in the right plot is the regression model fitted by switching the x/y values. Data from Kroah-Hartman.<sup>885</sup> [code](#)

Take as an example, an analysis by Kroah-Hartman<sup>885</sup> who measured the number of commits made between the release of a version of Linux and the immediately previous version, and the number of developers who contributed code to that release, for the 67 major kernel releases between versions 2.6.0 and 4.6.

In the upper plot of Figure 11.5, the number of developers is treated as the explanatory variable (x-axis) and number of commits as the response variable (y-axis), with the fitted regression line in red and dashed lines showing the difference between measurement and fitted model. In the lower plot the explanatory/response roles played by the two variables when fitting a regression model is switched; to simplify comparison the axis denote the same variables in both plots, with the green line denoting the newly fitted model and dashed lines showing the difference between measurement and model (now on the x-axis response variable; the line fitted in the upper plot is also plotted for comparison, still in red).

In the first case the fitted equation is:

$$\text{commits} = -237 \pm 523 + (8.7 \pm 0.44) \times \text{Number\_devs}$$

transforming this equation we get:

$$\begin{aligned}\text{Number\_devs} &= \frac{237 + \text{commits}}{8.7} \\ &= 27 + 0.11\text{commits}\end{aligned}$$

However, if a new model is fitted by switching the roles of the two variables, in the formula passed to `glm`, the returned model is described by the following equation:

$$\text{Number\_devs} = 162 \pm 52 + (0.10 \pm 0.005) \times \text{commits}$$

which is different from the equation obtained by transforming the first fitted model.

There is another difference between the two fitted models, the second model is a better fit to the data. Somebody only interested in the quality of fit might be tempted to select the second model, purely for this reason.

What is the procedure for deciding which measurement variables play the role of response and explanatory variable, e.g., should number of developers be considered an explanatory or response variable?

An important attribute of explanatory variable(s) is that their value is controlled by the person making the measurement. For instance, the model building process used to create Figure 11.2 had number of days as the explanatory variable; this choice was completely controlled by the person making the measurements.

The Kroah-Hartman commit measurements are based on the day of release of a version of the Linux kernel, a date that is outside the control of the measurement process. In fact both measurements have the characteristics of a response variables, that is, the value they have, was not selected by the person making the measurement. Both the possibility of variation in Linux version release dates and variation in number of commits made by developers are sources of uncertainty, both variables need to be treated as containing measurement error.

Building a regression model using explanatory variables containing measurement error can result in models containing biased and inconsistent values, as well as inflating the Type I error rate.<sup>221, 1418</sup>

There are a variety of regression modeling techniques that can take into account error in the explanatory variable. These techniques are sometimes known as *model II* linear regression techniques (model I being the case where there is no uncertainty in the explanatory variables), and also as *errors-in-variable models*, *total least-squares* or *latent variable models*; methods used go by names such as *major axis*, *standard major axis* and *ranged major axis*.

If all the variables used to build a model contain some amount of error, then it is necessary to decide how much error each variable contributes to the total error in the model. Some model II techniques are not scale invariant, that is they are only applicable if both axis are dimensionless or denote the same units, otherwise rescaling one axis (e.g., converting from kilometers to miles) will change its relative contribution. If each axis denotes a different unit, it does not make sense to use a model building technique that attempts to minimise some measure of combined uncertainty.

SIMEX (SIMulation-EXtrapolation) is a technique for handling uncertainty in explanatory variables that works in conjunction with a range of regression modeling techniques. The

SIMEX approach does not suffer from many of the theoretical problems that other techniques suffer from, but requires that the model builder provide an estimate of the likely error in the explanatory variable(s). The `simex` package implements this functionality, and supports a wide variety of regression models built by functions from various packages.

Continuing with the Linux developer/commit count example, to build a regression model using SIMEX, we need an estimate of the uncertainty in the number of developers contributing at least one commit to any given release. The `simex` function taking a model built using `glm` (and by other regression model building functions) and an estimate of the uncertainty in one or more of the explanatory variables, and returns an updated model that has been adjusted to take this uncertainty into account.

The following is a rough and ready approach to estimating the uncertainty in the Kernel attributes, measured by Kroah-Hartman:

- the release date of a new version of Linux is assumed to have an uncertainty of  $\pm 14$  days about the actual release date.<sup>ix</sup>
- the possible variation in the unique contributor count for any release is assumed to be uniformly distributed in the range: measured contributor count plus/minus number of developers contributing their first commit in the last 14 days.
- based on these assumptions, a standard deviation of 41 is obtained for the number of unique developers making at least one commit, averaged over all versions (see [regression/clean/dev-commit.R](#)).

Integrating this estimate of the standard deviation in the explanatory variable into a regression model is a two-step process:

- first build a regression model using `glm` in the usual way, but with the optional named parameter `x` set to `TRUE` (`y` also needs to be `TRUE`, but this is its default value and so the assignment below is redundant),
- pass the model returned by `glm` to `simex`, along with the name of the explanatory variable and its estimated standard deviation.

In code:

```
yx_line = glm(commits ~ developers, x=TRUE, y=TRUE)

sim_mod=simex(yx_line, SIMEXvariable="developers", measurement.error=41)
```

The following equation is derived from the model returned by `simex` (see [regression/dc-simex.R](#)):<sup>x</sup>

$$\text{commits} = -387 \pm 453 + (8.9 \pm 0.4) \times \text{Number\_devs}$$

The error in individual explanatory variable measurements can be specified by assigning a vector to `measurement.error` (the argument `asymptotic=FALSE` is also required); see Figure 11.34.

How reliable is a fitted model that ignores any uncertainty/error in explanatory variable measurements? The only way to answer this question is to build a model that takes this error into account and compare it with one that does not. The difference between the two ways of structuring fitted models can sometimes be much larger than that in Figure 11.5.

The question of whether economies of scale exist for software development, can be answered by analysing project effort/size data. Figure 11.6 shows lines for two fitted regression models, one with Effort as the explanatory variable the other with Size as the explanatory variable (from a study by Jørgensen, Indahl and Sjøberg<sup>803</sup>). If economies of scale exist, the slope of the effort/size line will be less than one (diseconomies of scale produce a slope greater than one). In this case, one slope is less than one and the other greater than one. The models fitted by switching response/explanatory variables are outside each other's 95% confidence intervals (there is no reason to expect them to be inside).

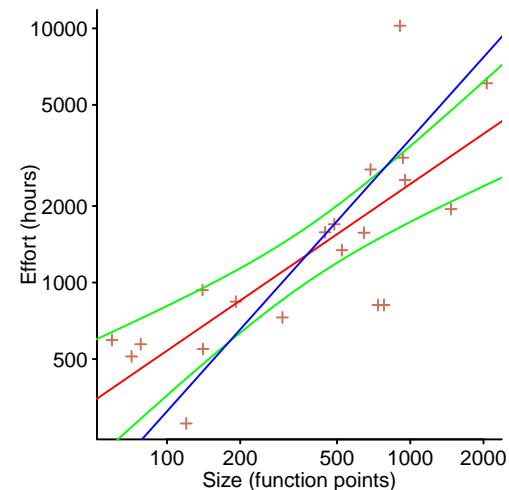


Figure 11.6: Effort/Size of various projects and regression lines fitted using Effort as the response variable (red, with green 95% confidence intervals) and Size as the response variable (blue). Data from Jørgensen et al. <sup>803</sup> [code](#)

<sup>ix</sup> Pointers to a more reliable, empirically derived, value are welcome.

<sup>x</sup> Readers might like to experiment with the value of the `measurement.error` to see the impact on the model coefficients.

Many of the measurement values treated as explanatory variables in this book were not under the control of the person who measured them. For instance, lines of code, number of files and reported problems measured at a given point in time are all response variables. To reduce your author's workload, most of the model fitting in this book does not make any adjustments for errors in the explanatory variables.

There are techniques, and R packages, for building complete models starting from the data, rather than refitting an existing regression model. For those wanting to build a model from scratch, the `lmodel2` package provides functions that implement many of the available methods.

### 11.2.4 Modeling data that curves

A model based on a straight line is a wonderful thing to behold, it is simple to explain and often aligns with people's expectations (many real world problems are well fitted by a straight line). However, life is complicated and throws curved data at us.

Having encountered an operating system having constant lines of code growth over many years, it is tempting to draw a conclusion about the growth rate of other operating systems. However, the way in which the data points curve around the fitted line in the upper plot of Figure 11.7, suggests that some of the processes driving the growth of the Linux kernel are different from those driving FreeBSD; perhaps a quadratic or exponential equation would be a better fit (these possibilities were chosen because they are two commonly occurring forms for upwardly curving data).

This section is about fitting linear models, so the possibility of an exponential fit is put to one side for the time being; building non-linear models, including better fitting non-linear models to this data, is discussed later (see Section 11.5).

The following call to `glm` fits an equation that is quadratic in the variable `Number_days`; the righthand side of the formula contains `Number_days+I(Number_days^2)`. The `I` (sometimes known as *as-is*) causes its argument to remain unevaluated and is treated as a distinct explanatory variable (the `^` operator has a distinct meaning within `glm`'s formula notation, see Table 11.2, and use of `I` prevents the expected binary operator usage being overridden). An alternative way of including a squared explanatory variable in the model, is to assign the value `Number_days^2` to a new variable and include this new variable's name on the righthand side of the formula. The result of fitting this equation is shown in the lower plot of Figure 11.7.

```
linux_mod=glm(sloc ~ Number_days+I(Number_days^2), data=linux_info)
```

The quadratic fit looks like it is better than linear, but perhaps a cubic, quartic or higher degree polynomial would be even better fits. The higher the order of the polynomial used, the smaller the error between the fitted model and the data. The error decreases because the additional terms make it possible to do a better job of following the random fluctuations in the data. A method of applying Occam's razor is needed, to select the number of terms that produces the simplest model consistent with the data and having an acceptable error.

The Akaike Information Criterion, AIC, is a commonly used metric for comparing two or more models (available in the `AIC` function). It takes into account both how well a model fits the data and the number of free coefficients in the model (i.e., constants selected by the model building process, such as polynomial coefficients); free coefficients have to pay their way by providing an appropriate improvement in a model's fit to the data.<sup>xi</sup> AIC can also be viewed as the information loss when the true model is not among those being considered.<sup>232</sup>

One set of selection criteria<sup>232</sup> are that models whose AIC differs by less than 2 are more or less equivalent, those that differ by between 4 and 7 are clearly distinguishable, while those differing by more than 10 are definitely different.

How much better does a quadratic equation fit Linux SLOC growth, compared to a straight line and how much better do higher degree polynomials fit? The following list gives the AIC for models fitted using polynomials of degree 1 to 4 (lower values of AIC are better). After initially decreasing the AIC starts to increase, once a fourth degree polynomial is reached; the third degree polynomial is thus the chosen linear polynomial, of those tested (other forms of equation could provide better fits using fewer free coefficients.) `code`

<sup>xi</sup> A negative AIC value may be the result of a nominal explanatory variable having many values, e.g., dates that are represented as strings, which could be converted using `as.Date`.

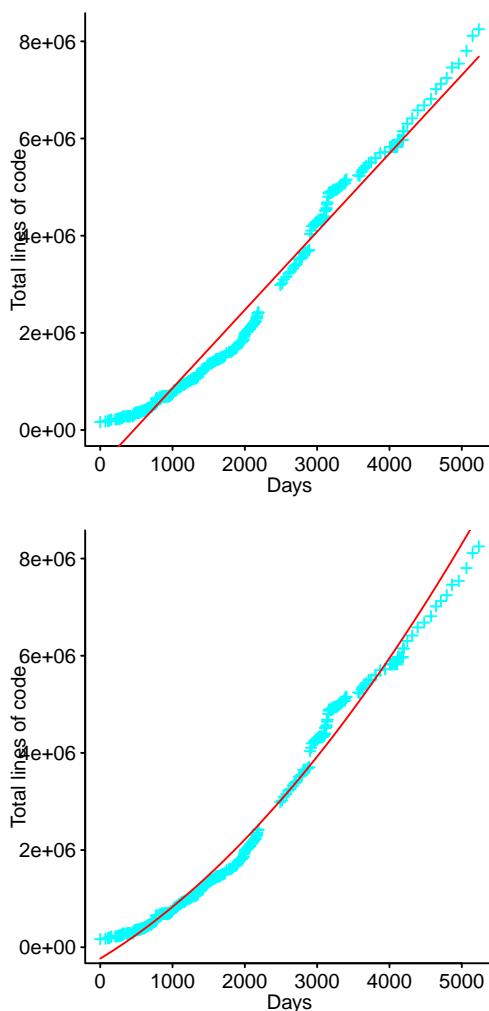


Figure 11.7: Lines of code in every initial release (i.e., excluding bug-fix versions of a release) of the Linux kernel since version 1.0, along with fitted straight line (upper) and quadratic (lower) regression models. Data from Israeli et al.<sup>762</sup> `code`

```
[1] Degree 1, AIC= 13998.0004739753
[1] Degree 2, AIC= 13674.6883243397
[1] Degree 3, AIC= 13220.8542892188
[1] Degree 4, AIC= 13221.7072389496
```

The following is the summary output of the fitted cubic model: [code](#)

Call:

```
glm(formula = LOC ~ Number_days + I(Number_days^2) + I(Number_days^3),
  data = latest_version)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-428217	-80061	6503	64889	620500

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	3.432e+05	2.876e+04	11.935	< 2e-16 ***
Number_days	-3.664e+02	5.144e+01	-7.123	3.79e-12 ***
I(Number_days^2)	8.167e-01	2.456e-02	33.258	< 2e-16 ***
I(Number_days^3)	-9.184e-05	3.371e-06	-27.242	< 2e-16 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 23001633959)

Null deviance: 1.8867e+15 on 494 degrees of freedom

Residual deviance: 1.1294e+13 on 491 degrees of freedom

AIC: 13221

Number of Fisher Scoring iterations: 2

Regression modeling produces the best fit for an equation over the range of data values used, and AIC helps prevent overfitting. No claims are made about how well the model is likely to fit data outside the range values used to fit it. Using a model, optimized to fit the available data, to make predictions outside the interval of the data values used can produce very surprising results.

What is the behavior of the above cubic model outside its fitted range, and in particular, what predictions does it make about future growth? Figure 11.8 shows that the model predicts a future decrease in the number of lines of code. A decreasing number of lines is the opposite of previous behavior, this prediction does not appear believable (if this decreasing behavior were predicted by a more detailed model of code growth, that closely mimicked real-world development by using information on the number of developers actively involved and a list of functionality likely to be implemented, it would be more believable).

A quadratic equation might not fit the data as well as a cubic equation, but the form of its predictions (increasing growth) is consistent with expectations.

If the purpose of modeling is understanding, then the quadratic model maps more closely to anticipated behavior; if the purpose is prediction within the interval of the fitted data, then the cubic model is likely to have a smaller error.

What about fitting other kinds of equations to the data? Equations such as  $Y = \alpha e^{\beta X} + \varepsilon$  and  $Y = \alpha X^\beta + \varepsilon$  are nonlinear in  $\beta$ ; non-linear model building is discussed in Section 11.5.

For a software system to grow, more code has to be added to it than is deleted. A constant rate of growth suggests either a constant amount of developer effort or a bottleneck holding things up; an increasing rate of growth (i.e., quadratic) suggests an increasing rate of effort. The different code growth pattern seen in the Linux kernel, compared to NetBSD/FreeBSD and various other applications, has been tracked down<sup>579</sup> to device driver development; new hardware devices often share many interface similarities with existing devices; for Linux developers tend to copy an existing driver, modifying it to handle the hardware differences. It is this reuse of existing code that is the source of what appears to be a non-linear growth in developer effort.

This method of creating a new device driver, performed by many developers working independently, can continue for as long as there are new devices coming to market; the evolution of Linux device drivers is discussed elsewhere...

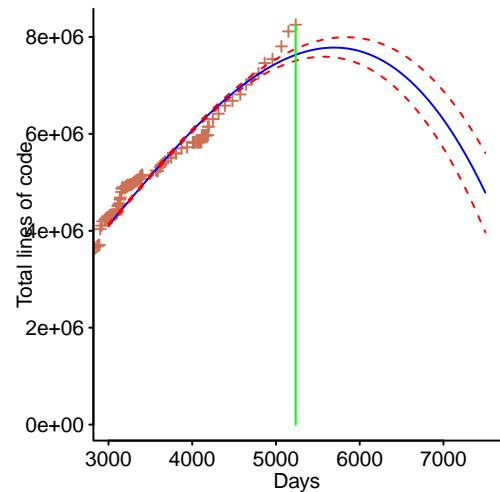


Figure 11.8: Actual (left of vertical line) and predicted (right of vertical line) total lines of code in Linux at a given number of days since the release of version 1.0, derived from a regression model built from fitting a cubic polynomial to the data (dashed lines are 95% confidence bounds). Data from Israeli et al.<sup>762</sup> [code](#)

A linear regression model is not restricted to combining explanatory variables using polynomials, any function can be used as long as the coefficients of the model occur in linear form. For instance, the FreeBSD model plotted in Figure 11.2 might include a seasonal term that varies with time of year; while a model containing the term  $A \sin(2\pi ft + \phi)$ <sup>xii</sup> is nonlinear (because of  $\phi$ , the phase shift), it can be written in linear form the follows:

$$A \sin(2\pi ft + \phi) = \alpha_s \sin(2\pi ft) + \alpha_c \cos(2\pi ft)$$

where:  $\alpha_s = A \cos \phi$  and  $\alpha_c = A \sin \phi$ ;  $A = \sqrt{\alpha_s^2 + \alpha_c^2}$  and  $\phi = \arctan \frac{\alpha_s}{\alpha_c}$ .

The call to `glm` is now (the argument to the trig functions is in radians):

```
rad_per_day=(2*pi)/365
freebsd$rad_Number_days=rad_per_day*freebsd$Number_days
season_mod=glm(sloc ~ Number_days+sin(rad_Number_days)+cos(rad_Number_days),
               data=freebsd)
```

The summary output, from the fitted model, shows that while a seasonal component exists, its overall contribution is small (see [regression/Herraiz-BSD-season.R](#)).

While fitting a model using all available measurements points is a reasonable first step, subsequent analysis may suggest that the data might best be treated as two or more disjoint samples. There may be time dependent factors that have a strong influence on growth patterns.

Figure 11.9 shows the number of classes in the Groovy compiler at each release, in days since version 1.0. There are noticeable kinks in the growth rate at around 1,300 and 1,500 days. Fitting a model to the complete sample, shows upward trending quadratic growth in the number of classes over time, but fitting separate models to two halves of the sample, shows quadratic growth that flattens out.

An investigation the Groovy compiler developer, finds that the kink occurs at a transition between version numbers. It is possible to invent a variety of explanations for the pattern of behavior seen, but treating the measurements as-if they came from a single continuously developed code base, is probably not one of them; further investigation of the circumstances behind the development of the Groovy compiler is needed, to obtain the desired level of confidence in one of these, or other, models.

## 11.2.5 Visualizing the general trend

Even when the measurement points are scattered in what appears to be a general direction, it is little work to confirm this general trend.

A general technique for highlighting the trend followed by data is to fit a regression model to a consecutive sequence of small intervals of the data, joining this sequence of fits together to form a continuous line. Two methods based on this idea (both fitting so the lines smoothly run together), are LOWESS (LOcally WEighted Scatterplot Smoothing) and LOESS (LOcal regrESSion); `lowess` and `loess` are the respective functions, with `loess` being used in this book.

A study by Kunst<sup>898</sup> counted, for 148 languages, the number of lines committed to Github (between February 2013 and July 2014) and the number of questions tagged with that language name on Stackoverflow.

The upper plot of Figure 11.10 shows lots of points which look as-if they trend in along straight line. The `loess` fit, red line in lower plot, shows the trend having a distinct curve. Experimenting with a quadratic equation in `log(lines_committed)` shows (blue line in lower plot) that this more closely follows the `loess` fit, than a straight line (a quadratic fit also has a lower AIC than a linear one; see [regression/langpop-corger-nl.R](#)).

A call to `loess` has the same pattern as a call to `glm`, with the possible addition of an extra argument; `span` is used to control the degree of smoothing:

```
loess_mod=loess(log(stackoverflow) ~ log_github, data=langpop, span=0.3)
x_points=1:max(langpop$log_github)
loess_pred=predict(loess_mod, newdata=data.frame(log_github=x_points))
lines(exp(x_points), exp(loess_pred), col=pal_col[1])
```

<sup>xii</sup> Some books use  $A \cos(2\pi ft + \phi)$ , which changes the phase by 90° and flips some signs.

A study by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner<sup>432</sup> investigated the effectiveness of web security code reviews, asking professional developers to locate vulnerabilities in code.

The lowess fit, blue line in Figure 11.11, suggests that the percentage of vulnerabilities found increases as the number of years working in security increases, but then rapidly decreases. This performance profile seems unrealistic. A fitted straight line, in red, shows a decreasing percentage with years of work in the security field (its p-value is 0.02).

Perhaps the correct interpretation of this data, is that average performance does increase with years worked in the field, but that the subjects with many years working in security, who took part in the study, were more managerial and customer oriented people (who had time available to take part in the experiment), i.e., this data contains sampling bias. At the time of the study, software security work was rapidly expanding, so the experience profile is likely to be skewed with more subjects being less experienced.

When it is not necessary to transform either argument, the value returned by the `loess.smooth` function can be passed directly to `lines`.

```
lines(loess.smooth(dev$experience, dev$written, span=0.5), col=pal_col[2])
```

A loess visualization can also be helpful when the number of data points is so large, they coalesce into formless blobs. The Ultimate Debian Database (see Figure 11.23) is an example.

The `loess` function divides the range of x-axis values into fixed intervals, which means that when the range of x-values varies by orders of magnitude, the fitted curve can appear over stretched at the low values and compressed at high values.

One solution is to reduce the range of x-values by, for instance, taking the log, smoothing and then expanding (see [regression/java-api-size.R](#)); the following code is used in Figure 11.31:

```
t=loess.smooth(log(API$Size), API$APIs, span=0.3)
lines(exp(t$x), t$y, col=loess_col)
```

## 11.2.6 Influential observations and Outliers

Influential observations are observations that have a disproportionate impact on the values of a model's fitted coefficients, e.g., a single observation significantly changes the slope of a fitted straight line. The terms *leverage* (or, based on the mathematical symbol used, *hat-value*) refer to the amount of influence a data point has on a fitted model; the `hatvalues` function takes the model returned by `glm` and returns the leverage of each point.

Influential observations might be removed or modified, or a regression technique used that reduces the weight given to what are otherwise overly influential points, e.g., the `glmrob` function in the `robustbase` package (which is not always as robust as desired and manual help may be required; see [regression/a174454-reg.R](#)).

Outliers are discussed as a general issue in Chapter 14, this subsection discusses outliers in the context of regression modeling. In this context an outlier might be defined as a data point having a disproportionately large standardized residual (here Studentized residuals are used). To repeat an important point made in that chapter: excluding any influential observations or outliers from the analysis is an important decision that needs to be documented in the results.

*Cook's distance* (also known as *Cook's D*) is a commonly used metric, which combines leverage and outlierness into a single number.

A study by Fenton, Neil, Marsh, Hearty, Radliński and Krause<sup>487</sup> involved data from 31 software systems for embedded consumer products. Figure 11.12 shows development effort against the number of lines of code, along with a fitted straight line and standard error bounds. At the right edge of the plot are two projects that consumed over 50,000 hours of effort, and the number of lines of code for these projects looks very small in comparison with other projects. Is the fitted model overly influenced by these two projects and should they be ignored or adjusted in some way?

As the number of points in a sample grows, there is an increasing probability that one or more of them will be some distance away from the fitted line; in any large sample a few apparent outliers are to be expected as a natural consequence of the distribution of the error.

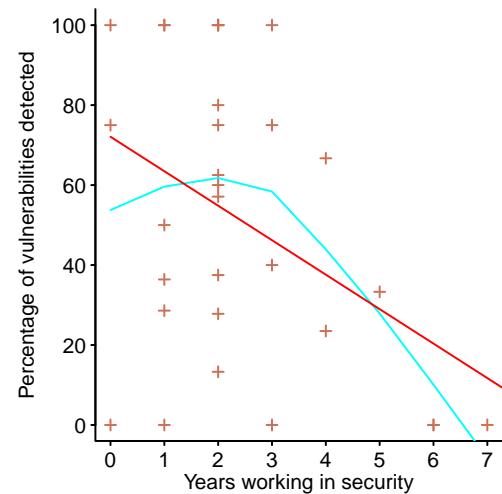


Figure 11.11: Percentage of vulnerabilities detected by developers who have worked a given number of years in security. Data extracted from Edmundson et al.<sup>432</sup> [code](#)

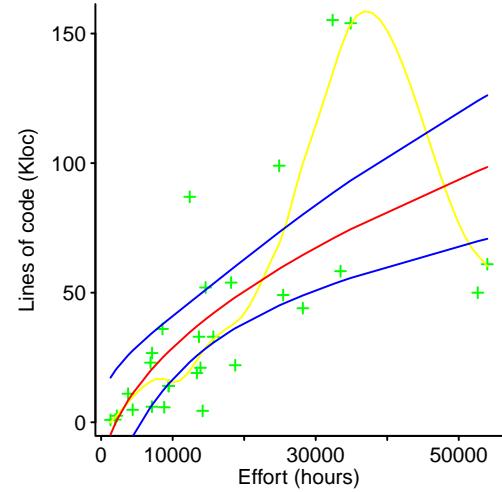


Figure 11.12: Hours to develop software for 29 embedded consumer products and the amount of code they contain, with fitted regression model and loess fit (yellow). Data from Fenton et al.<sup>487</sup> [code](#)

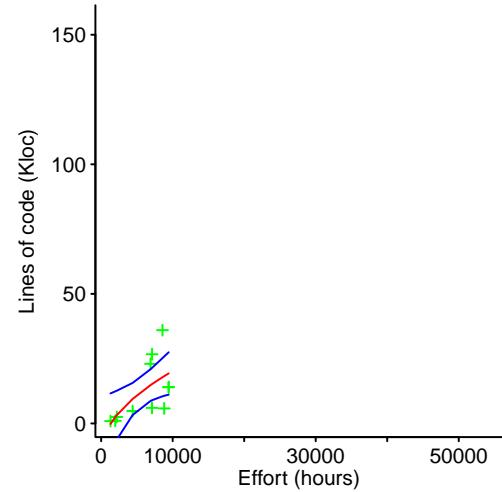


Figure 11.13: Points remaining after removal of overly influential observations, repeatedly applying Cook's distance and Studentized residuals. Data from Fenton et al.<sup>487</sup> [code](#)

The following analysis illustrates the dangers of not taking sample size into account, when making judgements about the outlier status of a measurement point.

Figure 11.13 shows the result of building a model, after removing measurements having both a high Cook's distance and Studentized residuals, and repeating the process until points stop being removed. At the end of the process, most of the measurement points have been removed.

Removing overly influential points until everything looks respectable is seductive, it is an easy to follow process that does not require much thought about the story that the data might have to tell. For those who don't want to think about their data, the `outlierTest` function in the `car` package can be used to automate outlier detection and removal (it takes a model returned by `glm` and returns the Studentized residuals of points whose Bonferroni corrected p-value is below a cutoff threshold; default cutoff=0.05).

A method of visualizing the important influential observation and outlier information is required. The `influenceIndexPlot` function in the `car` package, takes the model returned by `glm` and plots the Cook's distance, Studentized residual, Bonferroni corrected p-value and hat-value for each data-point; Figure 11.14 is for the Fenton et al data.

```
all_mod=glm(KLoC ~ I(Hours^0.5), data=loc_hour)
influenceIndexPlot(all_mod, main="", col=point_col, cex.axis=0.9, cex.lab=1.0)
```

The upper plot shows four data points having a large Cook's distance, but only two of them have a significant corrected p-value (second plot from the bottom). These two data points were removed and the process of building a model and calling `influenceIndexPlot` repeated; on this iteration one point is removed and iterating again finds no other data points as worthwhile candidates for removal.

Figure 11.15 shows the results of removing data points having both a high Cook's distance, and Studentized residuals whose corrected p-value is below the specified limit.

Outliers are loners, appearing randomly scattered within a plot. When multiple points appear to be following a different pattern than the rest of the data, the reason for this may be a different process driving behavior, or a change of behavior in what went before.

A study by Alemzadeh, Iyer, Kalbarczyk and Raman<sup>22</sup> investigated safety-critical computer failures in medical devices between 2006 and 2011 (as reported by the US Food and Drug Administration). Figure 11.16 shows the number of devices recalled for computer related problems (20-30% of all recalls), binned by two week intervals.

Points that stand out in the upper plot of Figure 11.16 are the two large recall rates in the middle of the measurement interval and recall rates at later dates appearing to increase faster than earlier; adding a loess fit (green) shows peaks around the two suspicious periods.

The fitted straight line shows a distinct upward trend. Is this fitted line being overly influenced by the two middle period points or end of measurement period recall rates?

The measurements points appear in regular time slots and deleting one of these time slots does not make sense; replacing an outlier with the mean of all measurements is one solution for handling this situation.

The lower plot shows regression lines (red original, purple after replacing two outliers with the mean), one of the peaks in the fitted loess line is reduced, but there is still a noticeable hump after 2010. There is little change in the fitted regression model, showing that these two outliers had little influence. Did a substantive change in the processes driving recalls, or recording of recalls, occur around the start of 2011? Further investigation, or domain knowledge, is needed to answer this question.

Figure 11.17 shows two fitted models, one using data up until the end of 2010 and the other using the data after 2010. This illustrates that blindly fitting a straight line to a sample can produce a misleading model. A change in reporting appears to have occurred around the end of 2010, which had a significant impact on reported recalls (work is needed to uncover the reason for this change) and fitting data up to the end of 2010 shows a much smaller increase, and perhaps even no increase, in recall rates, compared to when measurements after this date are included.

A change-point analysis of this data is discussed in Section 11.10. (see Figure 11.26).

When combining results from multiple studies, it is possible for an entire study to be an outlier, relative to the other related studies.

A study by Amiri and Padmanabhu<sup>37</sup> analysed the methods used by eleven other studies to convert between two common methods of counting function points.<sup>xiii</sup> Many of the studies included in the analysis have small sample sizes, include both student and commercial projects, and the function points are sometimes counted by academics rather than industrial developers.

Figure 11.18 shows function points counted using the COSMIC and FPA algorithms (counts made by students have been excluded). Both lines are loess fits, with red used for industry points and blue for academic researchers; the academic line overlays the industry line if one sample (i.e., Cuadtado\_2007) is excluded.

The impact of influential observations on a fitted model can vary enormously, depending on the form on the equation being fitted. Figure 11.19 shows the lines of five separate equations fitted to the Embedded subset of the COCOMO 81<sup>175</sup> data, with the upper plot using the original data and the lower plot the data after three influential observations have been removed.

In some cases outlier removal has had little impact on the model fitted, while in other cases there has been dramatic changes in the coefficients of the fitted model.

### 11.2.7 Diagnosing problems in a regression model

The commonly used regression modeling functions are capable of fitting a model to almost any sample, without reporting an error (some functions are so user-friendly they gracefully handle data that produces a singular matrix, an error that is traditionally flagged, because it suggests that something somewhere is wrong). It is the analysts' responsibility to diagnose any problems in the model returned.

Looking at Figure 11.20, it is immediately obvious that at least two of the fitted regression lines completely fail to capture the pattern present in the data. The data set is famous, it is known as the Anscombe quartet.<sup>49</sup> The four samples each contain two variables, with each sample having the same mean, standard deviation, Pearson correlation coefficient and are fitted using linear regression to produce a straight line having the same slope and intercept.

Problems with a regression model are not always as obvious as the Anscombe quartet case, and diagnosing the cause of the problem can be difficult. As always, domain knowledge is very useful for suggesting alternative models or possible changes to a fitted model.

The difference between the measured value of the response variable and the value predicted by a fitted model is known as the *residual*. While many model diagnosis techniques are based on the use of the residual, they often require more knowledge of the mathematics of regression modeling than is covered in this book.<sup>xiv</sup>

The suggested model diagnostic techniques, for casual users of statistics, are visualization based.

The upper plot in Figure 11.21 shows the residual of the straight line fitted to the Linux kernel growth data analysed earlier. Ideally the residual is randomly scattered around zero and the V-shape seen in this plot is typical of a straight line fitted to values that curve around it (the smallest residual is in the center, where the model fits best, and is greatest at the edges; the smaller peak is a localised change of behavior and may explain why a cubic produces a slightly better fit). This plot is one of the four diagnostic visualizations produced by `plot`, when it is passed a regression model, as follows:

```
m1=glm(LOC ~ Number_days, data=latest_version)
plot(m1, which=1, caption="", col=point_col)
```

The lower plot in Figure 11.21 shows the original data, straight line fit (red) and loess fit (blue). Both the residual plot and loess fit express the same pattern of curvature around about the straight line fit. Both visualizations have their advantage, the loess line can be drawn before any model is fitted, while details are easier to extract, from a residual plot (e.g., values for the size of the difference).

<sup>xiii</sup> Function point counting is a technique for estimating development effort by counting the functionality contained in the software requirements specification.

<sup>xiv</sup> It is not obvious that the cost/benefit of learning the necessary mathematics is worthwhile (but it is a good source of homework exercises for students).

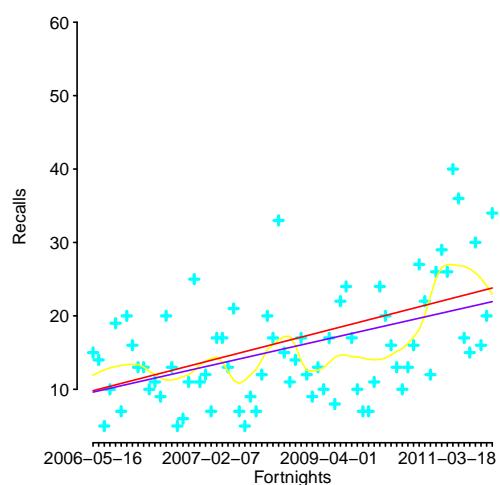
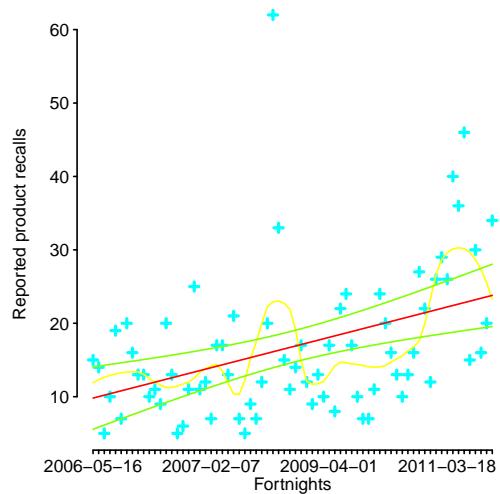


Figure 11.16: Number of medical devices reported recalled by the US Food and Drug Administration, in two week bins. Upper: fitted straight line and confidence bounds, with loess fit (yellow); Lower: straight line (purple) fitted after two outliers replaced by mean and original fit (red). Data from Alemzadeh et al.<sup>22</sup> code

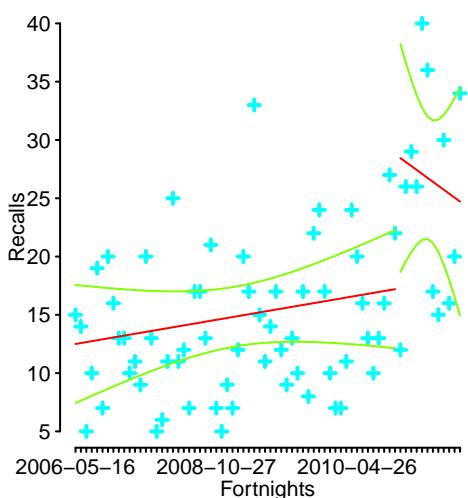


Figure 11.21: Two fitted straight lines and confidence intervals, one up to the end of 2010 and one after 2010. Data from Alemzadeh et al.<sup>22</sup> code

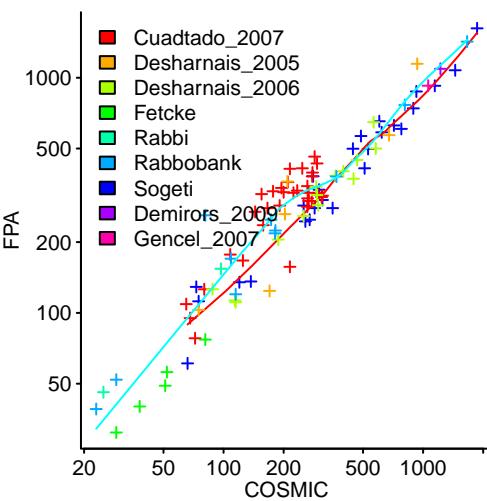


Figure 11.18: Results from various studies of software requirements function points counted using COSMIC and FPA; lines are loess fits to studies based on industry and academic counters. Data from Amiri et al.<sup>37</sup> [code](#)

The mathematics behind linear regression requires that each measurement be independent of all the other measurements in a sample. A common form of dependence between measurements is serial correlation, i.e., correlation between successive measurements. A fitted regression model can be tested for serial correlation using the `durbinWatsonTest` function, in the `car` package.

A study by Flater and Guthrie<sup>509</sup> measured the time taken to assign a value to an array element in C and C++, using twelve different techniques, some of which checked that the assignment was within the defined bounds of the array (two array sizes were used, large and small); the programs benchmarked were compiled using seven different compiler optimization options.

Figure 11.22 shows the timings from 2,000 executions of one technique for assigning to an array element, compiled using `gcc` with the `00` option (upper) and `03` option (lower). The results for `00` show a clustering of execution times for groups of successive measurements.

A Durbin Watson test confirms that the `00` measurements are correlated (see [benchmark/array-durbanwatson.R](#)).

When measurements contain a significant amount of serial correlation, the technique of time-series analysis, may be more appropriate; see Section 11.10. The `tsglm` function, in the `tscount` package, supports regression modeling of count time series.

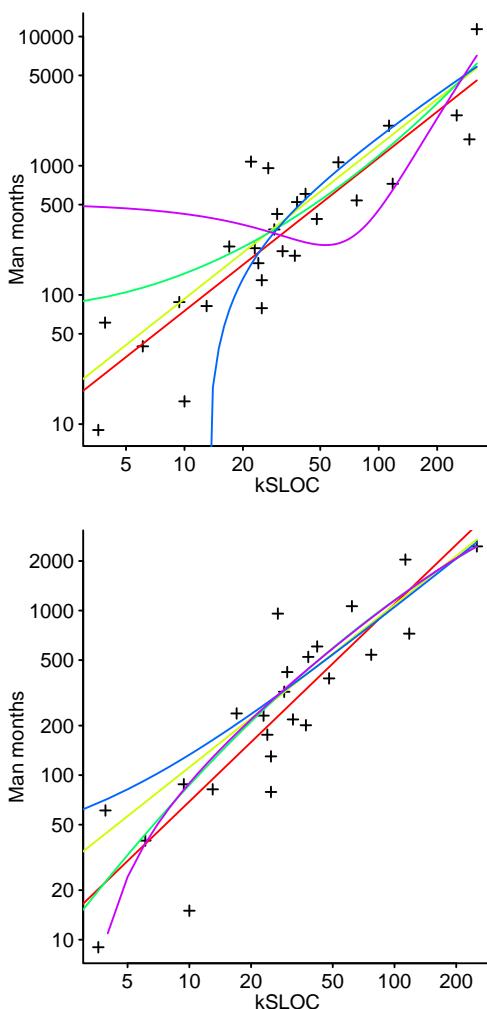


Figure 11.19: Five different equations fitted to the Embedded subset of the COCOMO 81 data before influential observation removal (upper) and after influential observation removal (lower). Data from Boehm.<sup>175</sup> [code](#)

## 11.2.8 A model's goodness of fit

How well does a model fit the data? The term *goodness of fit* is often used to describe this quantity. Various formula for calculating a goodness of fit have been proposed, and often involve the difference between the value measured and the corresponding value predicted by the model.

For the end-user of results of the analysis, meeting expectations of behavior is an important model characteristic.

When fitting equations to gain understanding, the structure of the processes suggested by the terms of the equation are an important characteristic.

When making predictions, the primary quantity of interest is the accuracy of new predictions, i.e., the amount of expected error in predictions for values that are not in the sample used to build the model. The error structure is also a consideration; is the priority to minimise total error, worse case error, to prefer over-estimates to under-estimates (or vice versa) or does some complicated weighting (over the range of values that explanatory variable(s) might take) have to be taken into account?

When dealing with one explanatory variable, it is possible to get a good idea of how well a model fits the data through visualization, e.g., by plotting them both. Does the fitted line look correct and how wide are the confidence intervals? However, for data containing more than one explanatory variable, accurate visualizations are problematic.

To create a model by fitting it to data, is to create a just so story. The predictions made by a model, outside the range of the data used to build it, are just something to discuss when considering expectations of behavior (which might be derived from a theory of the processes involved in generating the data used to fit the model).

Confidence intervals, see Figure 11.3, provide information about the goodness of fit at every point. The following discussion looks at some of the ways of producing a single numeric value, to represent goodness of fit.

The leftover variation in a sample that is not accounted for by the fitted model, the residual, is invariably a component in any equation in the calculation of a single value to summarise how well the model performs. Some of the equations used include:

- null deviance is a measure of the difference between the data and the mean of the data, deviance is a measure of the difference between the data and a fitted model (both values are listed in the summary output of a model fitted by `glm`). The percentage difference between the deviance and null deviance is a measure of the variance in the data that is not explained by the mode,
- R-squared (also known as the *coefficient of determination* and commonly written  $R^2$ ) can be interpreted as the amount of variance in the data (as measured by the residuals) that is

explained by a model. It takes values between zero and one (which has the advantage of being scale invariant) and is a measure of correlation, not accuracy.

Sometimes the adjusted  $R^2$ , written  $\bar{R}^2$ , is used, which takes into account the number of explanatory variables,  $p$ , and sample size,  $n$ :  $\bar{R}^2 = R^2 - (1 - R^2) \frac{p-1}{n-p}$

- mean squared error (MSE): the mean squared error is the mean value of the square of the residuals and as such has no upper bound (and will be heavily influenced by outliers); root mean squared error (RMSE) is the square-root of MSE.

The following equation shows how MSE and  $R^2$  are related:  $R^2 = 1 - \frac{MSE}{\sigma^2}$

- mean absolute error (MAE): the mean absolute error is the mean value of the absolute value of the residuals. This measure is more robust in the presence of outliers than MSE.

Apart from the  $R^2$  metric, the metrics listed (plus AIC) are scale dependent, e.g., mapping measurements from centimeters to inches changes their value; transforming the scale (e.g., taking logs) will also change metric values.

The choice of a metric is driven by what information is available and what model characteristics are considered important (e.g., how important is being able to handle outlier). In a competitive situation, people might not be willing to reveal details about their model and so any public metric has to be based on predictive accuracy (e.g., model builders provide the predictions made by their model to a test data set).

$R^2$  is the only scale invariant metric, and it provides an indication of how much improvement might be possible over an existing model.

It is possible for the coefficients of a fitted model to be known with a high degree of accuracy, and yet for this model to explain very little of the variance present in the data, and for there to appear to be little chance of improving on the model given the available data.

The Ultimate Debian Database project<sup>1578</sup> collects information about packages included in the Debian Linux distribution. Figure 11.23 shows the age of a packaged application plotted against the number of systems on which that application is installed, for 14,565 applications in the "wheezy" version of Debian. Also, see Figure 6.2.

The fitted linear model (red line, hidden by the 95% confidence interval in green overwriting it; loess fit in blue) has a very low p-value, a consequence of the large number of, and uniform distribution of, data points. The predictive accuracy of this model is almost non-existent, the only information it contains is that older packages are a little more likely to be installed than younger ones.

A study by Jørgensen and Sjøberg<sup>808</sup> investigated developers' ability to predict whether any major unexpected problems would occur during a software maintenance task. Building a regression model, using the available measured attributes, finds that lines of code is the only explanatory variable having a p-value less than 0.05. However, only 3.3% of the variance in the response variable is explained by the number of lines of code; while the explanatory variable was statistically significant, its practical significance was negligible (see [maintenance/10.1.1.37.38.R](#)).

## 11.2.9 Low signal-to-noise ratio

Measurements sometimes contain a large amount of noise, relative to the signal present, i.e., a low signal-to-noise ratio. Fitting a model to data having a low signal-to-noise ratio can be difficult, because many equations do an equally (not very) good job.

The two plots along the upper row in Figure 11.24, show data generated from a quadratic equation containing noise, along with two fitted models (blue and green lines). The equation used to generate the two sets of data was:

$$y = x^2 + K \times (5 + rnorm(length(x)))$$

where:  $K = 10^3$  (left column) and  $K = 10^2$  (right column).

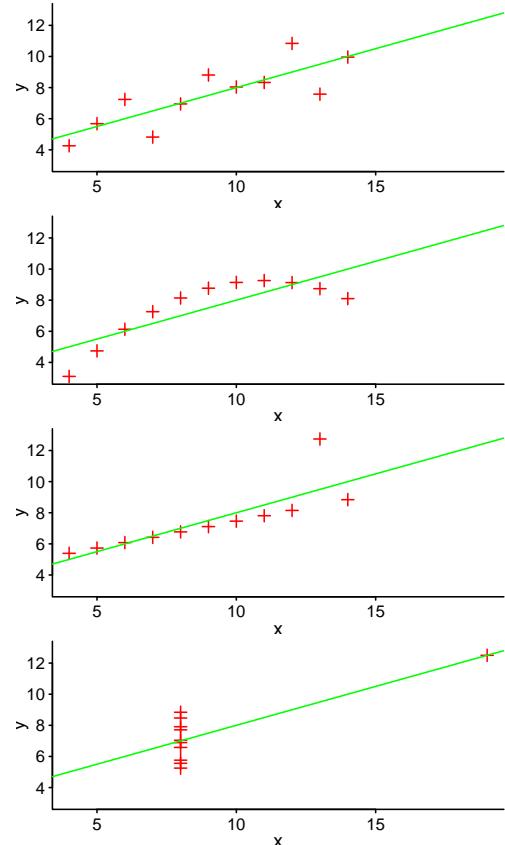


Figure 11.20: Anscombe data sets with Pearson correlation coefficient, mean, standard deviation, and line fitted using linear regression. Data from Anscombe.<sup>49</sup> [code](#)

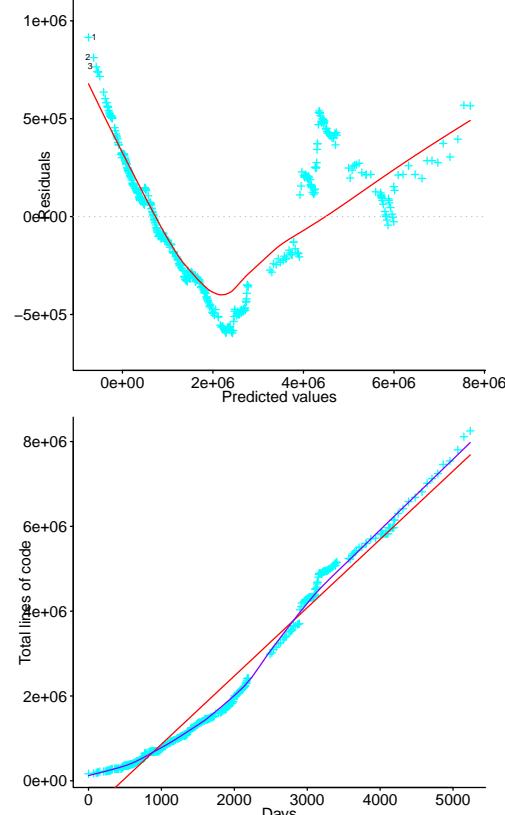


Figure 11.21: Residual of the straight line fit to the Linux growth data analysed in Figure 11.7 (upper) and data+straight line fit (red) and loess fit (blue). Data from Israeli et al.<sup>762</sup> [code](#)

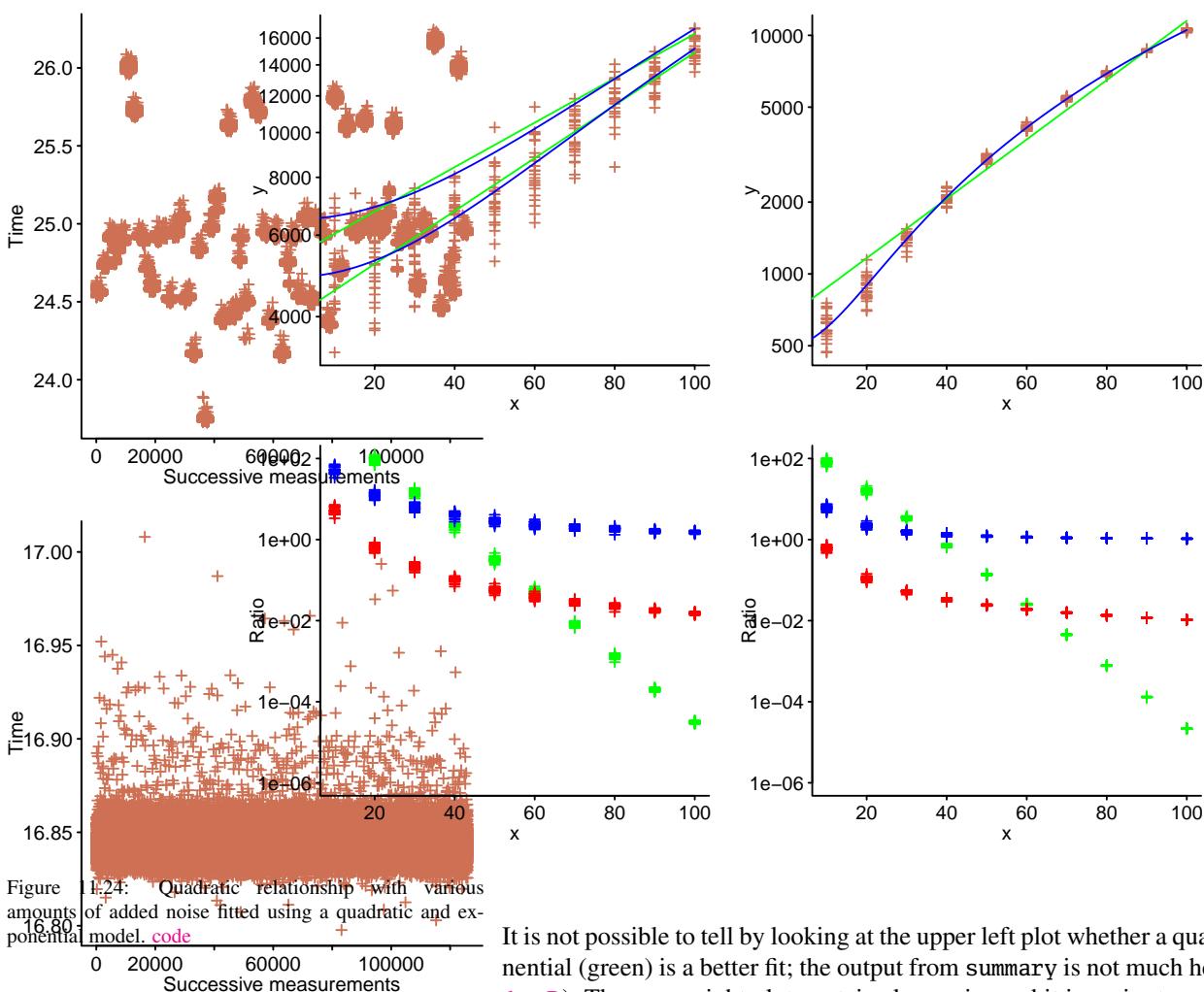


Figure 11.22: Array element assignment benchmark compiled with gcc using the 00 (upper) and 03 (lower) options (measurements were grouped into runs of 2,000 executions). Data from Flater et al.<sup>509</sup> [code](#)

Figure 11.22: Array element assignment benchmark compiled with gcc using the 00 (upper) and 03 (lower) options (measurements were grouped into runs of 2,000 executions). Data from Flater et al.<sup>509</sup> [code](#)

It is not possible to tell by looking at the upper left plot whether a quadratic (blue) or an exponential (green) is a better fit; the output from `summary` is not much help (see [regression/noisy-data.R](#)). The upper right plot contains less noise and it is easier to see that the exponential fit does not follow the data as well as the quadratic.

Sometimes the peaks (or troughs) in the plotted data can be an indicator of the shape of the data. The upper left plot includes a quadratic and exponential fit to the three largest values at each x-value (the fitted model does not seem to have less uncertainty in this case).

The *ratio test* is a technique that can help rule out some equations as possible candidates for modeling. If  $f(x)$  is the function being fitted to the data and this data was generated by the function  $g(x)$ , the ratio  $\frac{g(x)}{f(x)}$  will converge to a constant as  $x$  becomes small/large enough such that the signal dominates the noise.

The two plots along the lower row in Figure 11.24, show ratio tests for quadratic (blue), cubic (red) and exponential (green) equations. The exponential equation shows no sign of converging to a constant, while quadratic is closer to doing this than cubic (which can be ruled out because it does a poor job of fitting the data).

The ratio test rules out an exponential equation being a good candidate for fitting a model to the data in Figure 11.24.

A study by Vasilescu, Serebrenik, Goeminne and Mens<sup>1597</sup> investigated contributions to the Gnome ecosystem, from the point of view of workload (measured as the number of file touches, e.g., commits), breaking it down by projects, authors and number of activity types (e.g., coding, testing, documentation, etc).

The upper plot of Figure 11.25 shows, for individual authors, workload and the number of activity types they engaged in. There is a large amount of noise in the data (or variance not explained by the explanatory variable used for the x-axis). The lower plot of Figure 11.25 shows a ratio test, with an exponential failing to level off, the linear equation slowly growing and the quadratic looking like it is trying to grow.

Perhaps the behavior would become clearer with more activity types, but the quadratic is the only candidate not ruled out.

### 11.2.10 Abrupt changes in a sequence of values

When the processes generating the measured values changes, the statistical properties of the post-change sequence of values may abruptly change. The point at which the statistical properties of a value sequence significantly change is known as a *change-point*.

The `changepoint` package includes functions for performing change-point detection of the mean, variance and both mean and variance of a sequence of values. The `cpt.mean` function checks for significant shifts in the mean value; the `method="AMOC"` (At Most One Change) option searches for what its name implies; other values support searching for a specified maximum number of changes, with `method="PELT"` selecting what is considered to be the optimum number of changes.

An earlier analysis of electronic device recalls (see Figure 11.17) suggested that a significant shift in the processes driving recalls occurred at the end of 2010. Figure 11.26 shows the output from the following calls to `cpt.mean`:

```
library("changepoint")

change_at=cpt.mean(as.vector(t2))
plot(change_at, col=point_col,
     xlab="", ylab="Reported product recalls\n")

change_at=cpt.mean(as.vector(t2), method="PELT")
plot(change_at, col=point_col,
     xlab="Fortnights", ylab="Reported product recalls\n")
```

For an example of detecting changes in variance and changes in both mean and variance, see [regression/hpc-read-write.R](#).

The existence of a change-point is not always obvious, and when dealing with a few discrete measurement points change-point analysis might not be able to provide a reliable answer. As always, a loess curve can provide useful information.

A sequence of measurements containing an abrupt change could be fitted by multiple regression models, one each over each interval separated by a discontinuity. However, it is also possible to build a single model that contains the discontinuity information.

Figure 11.27 shows an abrupt change in the sales volume of 4-bit microprocessors (green). Straight lines have been fitted to the two periods before/after April 1998 (red), with the yearly sales cycle modeled by a sine wave (blue).

The technique for fitting a model that handles discontinuous patterns of behavior makes use of an interaction between the explanatory variable, date, and a dummy variable whose 0/1 value depends on date relative to the change-point. The code for the straight line model (line plotted in red) is:

```
# discontinuity point
y_1998=as.Date("01-04-1998", format="%d-%m-%Y")

p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998), data=proc_sales)
```

and the summary output is: [code](#)

```
Call:
glm(formula = bit.4 ~ date * (date < y_1998) + date * (date >=
y_1998), data = proc_sales)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-19756.9	-6372.8	-558.7	6533.2	19086.4

Coefficients: (2 not defined because of singularities)

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	7.050e+04	5.802e+04	1.215	0.2265
date	7.072e-01	5.368e+00	0.132	0.8954
date < y_1998TRUE	-8.357e+04	5.873e+04	-1.423	0.1572
date >= y_1998TRUE	NA	NA	NA	NA
date:date < y_1998TRUE	1.045e+01	5.466e+00	1.912	0.0581 .

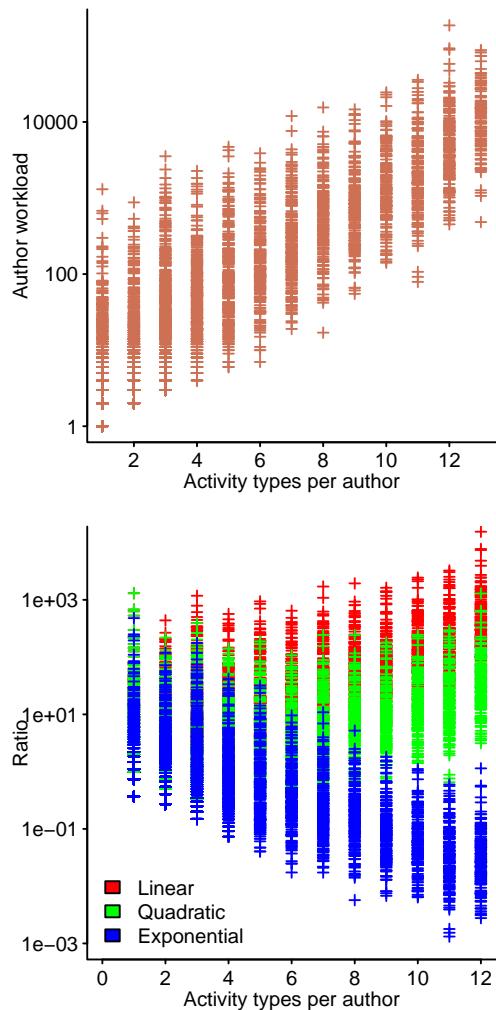
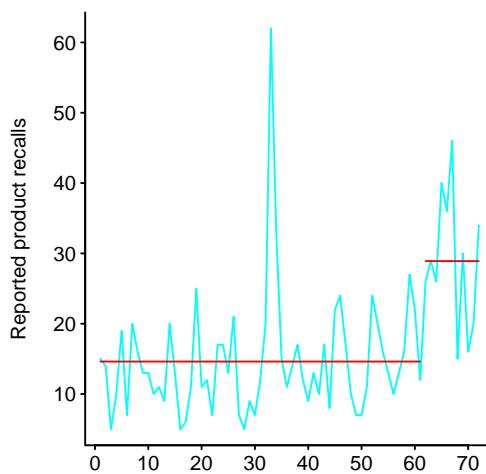


Figure 11.25: Author workload against number of activity types per author (upper) and ratio test (lower). Data from Vasilescu et al. [1597](#) code



```

date:date >= y_1998TRUE      NA      NA      NA      NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for gaussian family taken to be 80003408)

Null deviance: 2.0763e+10  on 131  degrees of freedom
Residual deviance: 1.0240e+10  on 128  degrees of freedom
AIC: 2782.6

Number of Fisher Scoring iterations: 2

Sales follow a seasonal trend that can be approximated using a sine wave having a 12-month frequency, adding this to the straight line model as follows:

season_p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998) +
sin(rad_days)+cos(rad_days), data=proc_sales)

```

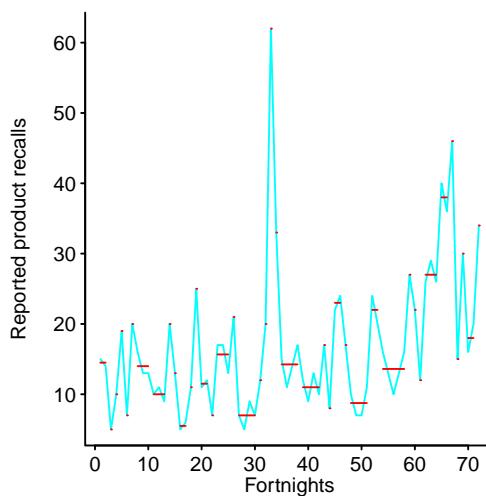


Figure 11.26: Change-points detected by `cpt.mean`, upper using method="AMOC" and lower using method="PELT". Data from Alemzadeh et al.<sup>22</sup> code

### 11.3 Moving beyond the default Normal error

Measurements sometimes have properties that do not meet the requirements assumed by the mathematics on which `glm`'s default argument values are based. Some of the measurement properties that non-default argument values can handle, include the response variable having values that:

- can never go below zero, e.g., count data,
- can never be greater than some maximum value, e.g., some percentages can never be greater than 100,
- span several orders of magnitude and contain an additive error.

By default, `glm` uses a Normal distribution for the measurement error. Figure 11.28 shows a fitted regression line with four of the data points (red stars adjacent to a black line); the colored Normal curves over each point represents the probability distribution of the measurement error that is assumed to have occurred for that measurement (the center of each error distribution curve is directly above the fitted line at each explanatory variable measurement point).

`glm`'s `family` argument has the default value `family=gaussian(link="identity")`, which can be shortened to `family=gaussian` (the default link function for `gaussian` is `link="identity"`).

The Normal distribution includes negative values and when a measurement cannot have a negative value, using an error distribution that includes negative values can distort the fitted model. One alternative is the Poisson distribution, which is zero for all negative values. The following call to `glm` specifies that the measurement error has a Poisson distribution:

```
a_model=glm(a_count ~ x_measure, data=some_data, family=poisson)
```

After Normal, the Poisson and Beta distributions are the most common measurement error distributions used by the analysis in this book.

Calling `glm`, with a non-default value for the `family` argument, requires knowing something about the mathematics behind generalised regression model building. The equation actually being fitted by `glm` is:

$$l(y + \varepsilon) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

which differs from the one given at the start of the chapter in having  $l(y + \varepsilon)$  on the left-hand-side, rather than  $y$ . This  $l$  is known as the *link function*, which for the Normal distribution is the identity function (this leaves its argument unmodified, and the equation ends up looking like the one given at the start of this chapter).

Once a regression model is fitted, the value of the response variable is calculated from:

$$y = l^{-1}(\alpha + \beta_1 x_1 + \beta_2 x_2 + \dots) - \varepsilon$$

where:  $l^{-1}$  is the inverse of the link function used, e.g., the inverse of log is  $e$  raised to the appropriate power.

Every error distribution has what is known as a *canonical link* function, which is the function that pops out of the mathematical analysis for that distribution. By default, `glm` uses the canonical link function for each error distribution, and allows some alternatives to be specified. The canonical link function for the Poisson distribution is `log`.

When the link function is not identity, prediction values and confidence intervals need to be mapped as follows

```
a_pred=predict(a_model, se.fit=TRUE)
inv_link=family(a_model)$linkinv      # get the inverse link function

lines(x_values, inv_link(a_pred$fit)) # fitted line
# confidence interval above and below
lines(x_values, inv_link(a_pred$fit+1.96*a_pred$se.fit))
lines(x_values, inv_link(a_pred$fit-1.96*a_pred$se.fit))
```

The analysis in the following sections involve measurements that require the use of a variety of measurement error distributions and link functions.

### 11.3.1 Count data

Count data has two defining characteristics, it is discrete and has a lower bound of zero. The discrete distribution taking on non-negative values, supported by `glm`, is the Poisson distribution.

In practice, when measurement values are sufficiently far away from zero (where far may be more than 10) there is little difference between models fitted using the Normal and Poisson distributions. For measurements close to zero, the main difference between models fitted using different distributions is the confidence intervals (which are usually not symmetric and may be larger/smaller).

The canonical link function for the Poisson distribution is `log` and the following two calls to `glm` are equivalent:

```
p_mod=glm(y ~ x, data=sample, family=poisson)
p_mod=glm(y ~ x, data=sample, family=poisson(link="log"))
```

The `log` link function means that the equation being fitted is actually:

$$y = e^{\alpha + \beta x} + \varepsilon$$

To fit the equation:  $y = \alpha + \beta x + \varepsilon$ , for a Poisson error distribution, the `identity` link function has to be used, as follows (experience shows that `glm` sometimes fails to converge when `family=poisson(link="identity")` is specified and that `start` values have to be specified):

```
p_mod=glm(y ~ x, data=sample, family=poisson(link="identity"))
```

A study of the effectiveness of security code reviews by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner<sup>432</sup> asked professional developers, with web security review experience, to locate vulnerabilities in web code. The number of vulnerabilities found can only be a non-negative integer value and in this study were single digit values.

The values fitted to a discrete distribution consists of a series of discrete steps, as the upper plot of Figure 11.29 shows (fitted line and 95% confidence intervals). While this plot is technically correct, it is ambiguous: are the values specified by the top left edge or the bottom right edge of the staircase?<sup>xv</sup> Plots using continuous lines are simpler for readers to interpret and so are used in this book.

The smooth lines in the lower plot of Figure 11.29 were fitted using the argument `family=poisson(link="identity")`, while the dashed lines were fitted using `glm`'s default values.<sup>xvi</sup>

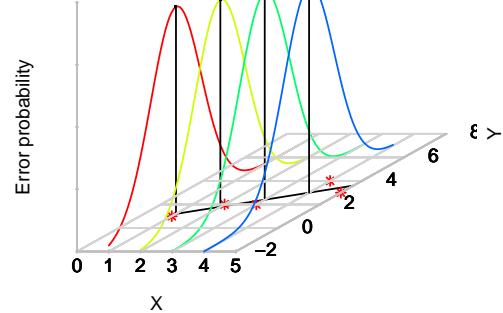


Figure 11.28: Fitted regression line to points (in red) and 3-D representation of assumed Normal distribution for measurement error. `code`

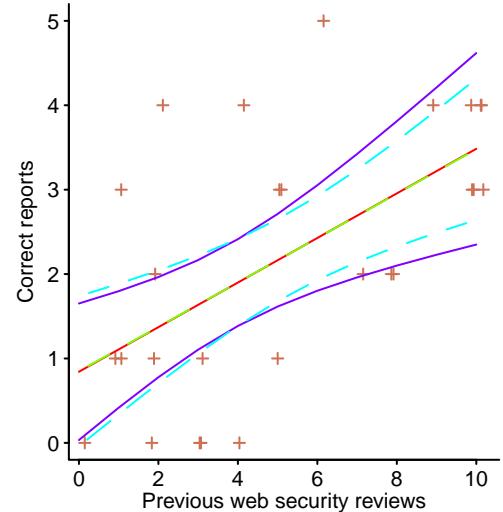
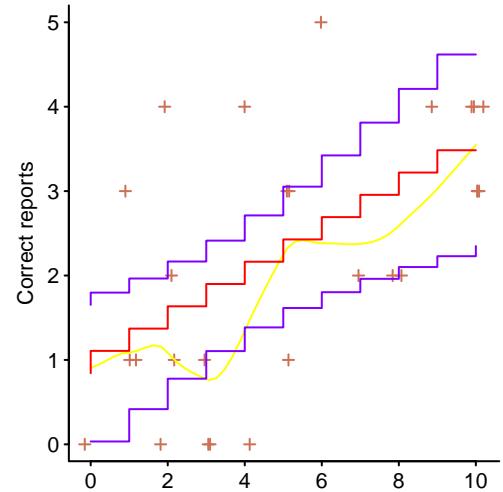


Figure 11.29: Number of vulnerabilities detected by professional developers with web security review experience; upper: technically correct plot of model fitted using a Poisson distribution, lower: simpler to interpret curve representation of fitted regression models assuming measurement error has a Poisson distribution (continuous lines) or a Normal distribution (dashed lines). Data extracted from Edmundson.<sup>432</sup> `code`

<sup>xv</sup> The choice is selectable via the `type` argument to `plot/lines`.

<sup>xvi</sup> To achieve an acceptable p-value, three outliers were removed.

The two fitted lines are virtually identical (the green dashed line is drawn over the continuous red line), but the 95% confidence intervals do differ. This pattern of behavior is very common, unless the response variable has many values near zero.

Is the difference between fitting a model using the technically correct Poisson distribution, or a Normal distribution, worth the effort (for the analyst, not the use of any additional computing resources)?

Sometime the Poisson distribution is used because a log link function transforms the response variable, while keeping an additive measurement error.

When fitting models containing multiple explanatory variables (discussed later) and a response variable containing count data, it can be more difficult to detect differences between using a Poisson and Normal distribution. While use of the Poisson distribution may involve more effort, it removes uncertainty and is always worth trying.

The Negative Binomial distribution is perhaps the second most commonly encountered count distribution. A study by Jones<sup>785</sup> included counting the number of break statements in C functions. Figure 11.30 shows the number of functions containing a given number of break statements, along with a fitted Negative Binomial distribution.

A break statement can occur zero or more times within a loop or switch statement, and these statements can occur zero or more times within a function definition. A Negative Binomial distribution can be generated by drawing values from multiple Poisson distributions (whose characteristics have been drawn from a Gamma distribution); might the number of break statements in each function different Poisson distribution?

The `gamlss` package<sup>1494</sup> supports a wide variety of probability distributions, including the NBI distribution (Negative binomial type I distribution; there is also a type II) used in the following code:

```
library("gamlss")
breaks=rep(j_brk$occur, j_brk$breaks)
nbi_bmod=gamlss(breaks ~ 1, family=NBI)

plot(function(y) max(jumps$breaks, na.rm=TRUE)*      # Scale probability distribution
      dnBI(y, mu=exp(coef(nbi_bmod, what="mu")),
            sigma=exp(coef(nbi_bmod, what="sigma"))),
      from=0, to=30, log="y", col=pal_col[1],
      xlab="breaks", ylab="Function definitions\n")
points(jumps$occur, jumps$breaks, col=pal_col[2])
```

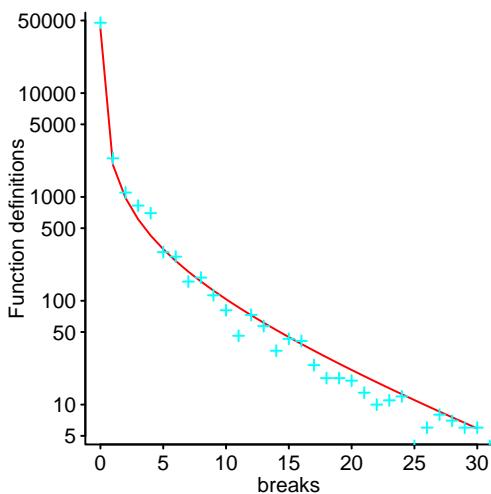


Figure 11.30: Number of functions containing a given number of break statements and a fitted Negative Binomial distribution. Data from Jones.<sup>785</sup> code

While zero is a common lower bound, other lower bounds are sometimes encountered; see Section 9.3.1. Both the `gamlss.tr` and `VGAM` packages support a wide variety of truncated distributions; `gamlss` and related packages are used in this book because of the volume and quality of their documentation.

A study by Starek<sup>1491</sup> investigated API usage in Java programs. Figure 11.31 shows the number of APIs used in Java programs containing a given number of lines of code. The API count starts at one, not zero, and many programs use a few APIs, suggesting that a Poisson distribution may be applicable; the range of the number of APIs used does not suggest a log scale.

In the following code, `gen.trun` creates a zero-truncated Poisson distribution (derived from `P0` in the `gamlss.tr` package) having the identity function as the link for its mean (rather than the default log link).

```
library("gamlss")
library("gamlss.tr")

gen.trun(par=0, family=P0(mu.link=identity))

tr_mod=gamlss(APIS ~ l_size+I(l_size^2), data=API, family=P0tr)
```

Figure 11.31: Number of APIs used in Java programs containing a given number of LOC; lines are fitted models based on a zero-truncated Poisson (red), Poisson and Normal distributions, yellow line is loess fit. Data from Starek.<sup>1491</sup> code

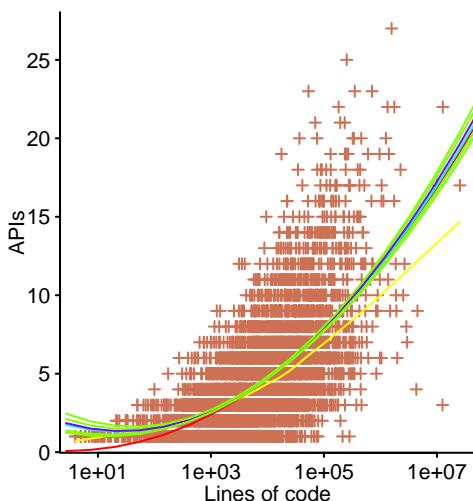


Figure 11.31 shows the fitted model in red. The other lines are fitted models using a Poisson distribution that is not zero-truncated and a Normal distribution, along with 95% confidence intervals. The yellow line is a loess fit. Other explanatory variables could be added to the model to improve the fit to the data.

### 11.3.2 Continuous response variable having a lower bound

Measurements of the response variable may be drawn from a continuous distribution, e.g., measurements involving length or time. The continuous distribution taking non-negative values, supported by `glm`, is the Gamma distribution.

In practice, when most measurement values are sufficiently far away from zero (where far away could be a large single digit value) there is little difference between models fitted using the Normal and Gamma distributions. For measurements closer to zero the main difference between models fitted using different distributions is in the confidence intervals (which are usually not symmetric and may be larger/smaller).

The canonical link function for the Gamma distribution is `inverse` and the following two calls are equivalent:

```
G_mod=glm(y ~ x, data=sample, family=Gamma) # Yes, capital G
G_mod=glm(y ~ x, data=sample, family=Gamma(link="inverse"))
```

The `inverse` link function means that the equation being fitted is (the `identity` link function is supported):

$$y = \frac{1}{\alpha + \beta x} + \epsilon$$

Figure 11.32 comes from a code review study (discussed in Section 13.2) and shows meeting duration when reviewing various amounts of code. Meeting duration must be greater than zero and a Gamma measurement error distribution is assumed to apply (the variables are assumed to have a linear relationship and the identity link function is used). The red line is the model fitted using a Gamma error distribution (plus confidence bounds), the green line is the Gaussian distribution fit.

The data contains a few points with high leverage and the loess fit suggests that there may be a change-point, so a more involved analysis is appears necessary.

### 11.3.3 Transforming the response variable

When plotting sample points, values along one or both axes are sometimes transformed to compress or spread out the points, for the purpose of improving data visualization.

A regression model is fitted to a pattern (represented by an equation) and if a plot using transformed axis, contains visible pattern(s) of behavior, it is worth investigating a model that uses similarly transformed values.

Applying a non-linear transform to the response variable changes its error distribution and a regression model built using this transformed response variable may not be a natural fit to the processes that produced the measurements. Explanatory variables are assumed not to contain any error and transforming them does not change this assumption.

For example, in the following regression model the error,  $\epsilon$ , is additive:

$$y = \alpha + \beta x + \epsilon$$

while fitting a log-transformed response variable:

$$\log y = \alpha + \beta x + \epsilon$$

produces a model where the error is multiplicative, i.e., the error present is a percentage of the measured value:

$$y = e^{\alpha + \beta x} e^\epsilon$$

The error in a model fitted using a log link function is additive, because the equation fitted is:

$$\log(y + \epsilon) = \alpha + \beta x$$

which becomes (the error randomly fluctuates around zero and negating it changes nothing):

$$y = e^{\alpha + \beta x} + \epsilon$$

If the response variable is transformed, the decision on whether to transform it directly or via a link function is driven by whether the error is thought to be additive or multiplicative; as always, domain knowledge is crucial.

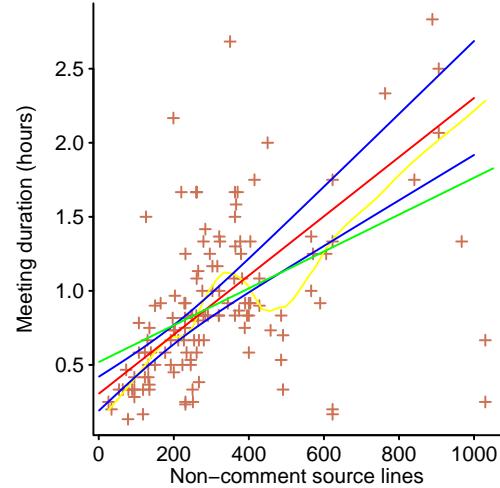


Figure 11.32: Code review meeting duration for a given number of non-comment lines of code; fitted regression model, assuming errors have a Gamma distribution (red, with confidence interval in blue) or a Normal distribution (green). Data from Porter et al.<sup>1264</sup> [code](#)

A log link can be specified for `glm`'s default Normal distribution by passing: `family=gaussian(link="log")`. If this use fails to converge, the Poisson distribution is a good approximation to the Normal distribution (except when many sample values are close to zero) and can be substituted when the response variable takes integer values (see Figure 7.6).

One advantage of log transforming a response variable is that it reduces the influence of outliers (because the range of values is compressed). Figure 11.33 illustrates the impact of removing one highly influential value from the data used to fit a model using a log link function (green lines are before/after) and a model fitted using a log transformed response variable (red lines are before/after);<sup>xvii</sup> the vertical shift is the difference between treating measurement error as additive and multiplicative.

The visual appearance of outliers and influential observations plotted using log axis can be deceiving, i.e., they may not appear to be that far removed from the general trend (see Figure 8.39). As always, assumptions based on visual appearance need to be checked numerically.

Many data analysts continue to fixate on fitting data whose measurement error has a Normal distribution. The Box-Cox transformation continues to be used to map a response variable to have a more Normal distribution-like error. The `boxcox` function in the `MASS` package and the `powerTransform` function in the `car` package provide support for this functionality.

A traditional approach to simplifying a problem is to map a continuous variable to a number of discrete values (e.g., small/medium/large). Throwing away information may simplify a problem, but the cost can be a considerable loss of statistical power and residual confounding.<sup>1358</sup> Using a computer removes the need to simplify just to reduce the manual effort needed to perform the analysis. See `regression/melton-statics.R` for an example where building a regression model provides a lot more information about the characteristics of the continuous data, compared to mapping values to large/small and running a chi-squared test.

Adjusting a fitted model to handle uncertainty in the explanatory variables, when the model contains a multiplicative error, requires specifying the measurement error for every value of an explanatory variable. The following option assigns a 10% error: `measurement.error=maint$lins_up/10`.

A study by Jørgensen<sup>796</sup> investigated maintenance tasks and obtained developer effort and code change data. Figure 11.34 shows the effort (in days) and number of lines inserted and updated for 89 maintenance tasks. The original fitted regression line is in red and the SIMEX adjusted line is in blue. The call to `simex` is:

```
maint_mod=glm(EFFORT ~ lins_up, data=maint,
              family=gaussian(link="log"), x=TRUE, y=TRUE)

y_err=simex(maint_mod, SIMEXvariable="lins_up",
            measurement.error=maint$lins_up/10, asymptotic=FALSE)
```

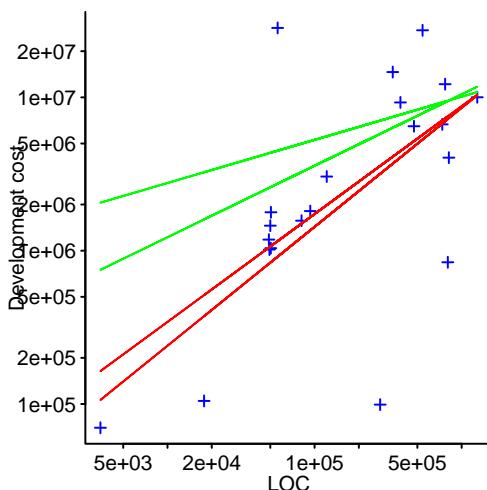


Figure 11.33: Annual development cost and lines of Fortran code delivered to the US Air Force between 1962 and 1984; lines show fitted regression models before/after outlier removed. Data extracted from NeSmith.<sup>1145</sup> [code](#)

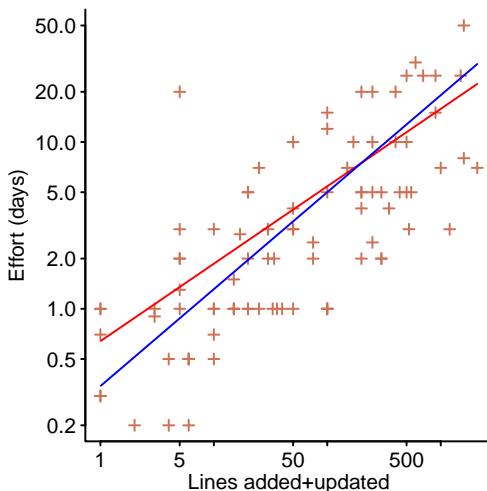


Figure 11.34: Maintenance task effort and lines of code added+updated, with fitted regression model (red) and SIMEX adjusted for 10% error (blue). Data from Jørgensen.<sup>796</sup> [code](#)

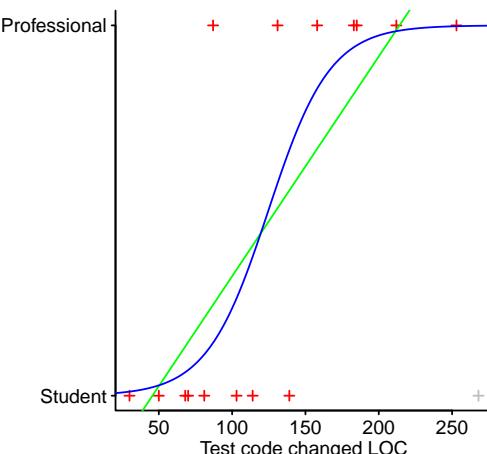


Figure 11.35: Regression modeling 0/1 data with a straight line and a logistic equation. [code](#)

### 11.3.4 Binary response variable

When the response variable takes one of two possible values, e.g., (false, true) or (0, 1), it has a binomial distribution. If the value of the response variable switches from 0 to 1 (or 1 to 0), as the explanatory variable increases (or decreases), and then always has that value for further increases (decreases) in the explanatory variable, there is no need to build a regression model (simply find the switch point). When the response variable can have two possible values over some range of the explanatory variable, regression modeling fits an equation that minimises some metric for the residual error.

A study by Höfer<sup>708</sup> investigated the various aspects of the implementing a problem by students and professional developers (working in pairs). Figure 11.35 shows the number of lines of test code changed by students and professionals (measurement denoted by the grey plus is treated as an outlier and not included in the model building), along with fitted regression lines, a straight line and a logistic equation.

(For an analysis of Microsoft's C/C++ compiler price differential under MSDOS and Windows, see `economics/upgrade-languages.R`)

The canonical link function for the Binomial distribution is `logit` and the following two calls are equivalent:

<sup>xvii</sup> The visual difference is less dramatic if the axes are switched.

```
b_mod=glm(y ~ x, data=sample, family=binomial)
b_mod=glm(y ~ x, data=sample, family=binomial(link="logit"))
```

The equation for the logit link function is:

$$\log \frac{y}{1-y} = \alpha + \beta x$$

where the response has the form of a log-odds ratio. This equation can also be written as:

$$\log \frac{p}{q} = \alpha + \beta x$$

where:  $p$  proportion of successes,  $q$  proportion of failures.

$$p = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$

The values returned by `predict`, for a fitted binomial model, are in the range 0...1. The analyst has to decide the point value that divides this continuous range, such that predictions are either zero or one. One approach is to treat predicted value greater than 0.5 as predicting one and predicted values less than or equal to 0.5 as predicting zero. A more sophisticated approach looks at the distribution of predictions, and makes an informed trade-off between true positive (in this context also known as *recall* and *hit rate*) and accuracy (i.e., the false positive rate).

A ROC curve (receiver operating characteristics; named after a technique used to measure the performance of radio receivers) is a visualization technique for showing the trade-offs between two rates, i.e., the true positive rate and false positive rate; it is a common technique for displaying the trade-offs from predictions returned by machine learning models. The ROCR package supports the creation and plotting of ROC curves.

The columns in Table 11.1 show an example of the impact of selecting particular cut-off values, for distinguishing between true/false (for 10 data points). Reading left-to-right, at a cut-point of 0.9 there is one correct prediction (a true positive), at a 0.81 cut-point another correct prediction, while at 0.72 an incorrect prediction (a false positive) is made (at this cut-point the response rate for correct predictions is 40% and 20% for incorrect predictions).

t	t	f	t	f	t	f	t	f	f
0.90	0.81	0.72	0.60	0.53	0.44	0.39	0.28	0.16	0.09

Table 11.1: Example list of prediction outcome occurring at various cut-point values. [code](#)

Figure 11.36 shows the ROC curve for this data.

AUC (Area Under the Curve) and select a cut-off value that provides the best trade-off between false-positive and....

In Figure 6.28... [reliability/fuzzer/fuzzer-mod.R](#) is a relatively complicated model... need something simpler...

### 11.3.5 Multinomial data

When discrete response variables take on more than two values, they have a *multinomial* distribution.

**nominal:** when a response variable can take  $N$  distinct values and  $\pi_i$  is the probability of the  $i^{th}$  value occurring ( $\sum_{i=1}^N \pi_i = 1$ ), then the *baseline-category logit model* (with one explanatory variable,  $x$ , in this example) is:

$$\log \frac{\pi_n}{\pi_N} = \alpha_n + \beta_i x$$

for  $n = 1, \dots, N - 1$ . Fitting a model results in  $N - 1$  equations, with separate coefficients for each.

The `mlogit` package supports the building of multinomial logit models for response variables containing nominal data.

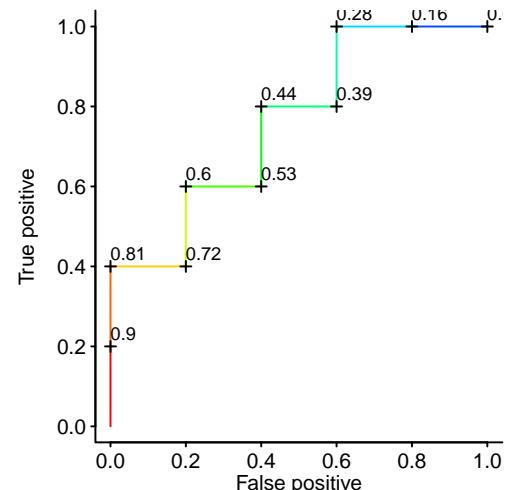


Figure 11.36: ROC curve for the data listed in Table 11.1. [code](#)

**ordinal:** fitting an independent logit model to each pair of adjacent values fails to make use of all the available information; the logit equation can be extended to include the ordering information present in ordinal data.

Given an ordinal response variable,  $Y$ , that can appear in one of  $j = 1, \dots, N$  possible categories, then  $Y$  has a multinomial distribution; its cumulative probability is given by:

$$P(Y_i \leq j) = \pi_{i1} + \pi_{i2} + \dots + \pi_{iN},$$

where:  $\pi_{ij}$  is the probability that the  $i^{\text{th}}$  measurement appears in response category  $j$ , and  $\sum_{j=1}^N \pi_{ij} = 1$

The *cumulative logits* treats the  $P(Y \leq j)$  as the response variable in a model fitting process that uses a logit link function (other, related functions can be used).

The *ordinal* package supports the building of *cumulative link models*, also known as *ordinal regression models*.

A study by Luthiger and Jungwirth<sup>992</sup> investigated the importance of fun as a motivation for software development. The survey, which had 1,330 responses from people working on open source projects, asked for an estimate of the percentage of their spare time people spent on activities involving open source development. Possible answers were restricted to intervals of 10%, an ordinal scale. The `clm` function fits a cumulative link model, however, note that `predict` returns a vector of predictions, for each of the ordinal values (six vectors in this example):

```
library("ordinal")
f_mod=clm(q42 ~ q31, data=fasd) # Best model is q42 ~ q5+q29+q31
pred=predict(f_mod, newdata=data.frame(q31=1:6))
plot(-1, type="n", xlim=c(1, 6), ylim=c(0, 0.6),
      xlab="q31", ylab="Probability\n")
dummy=sapply(1:10, function(X) lines(1:6, pred$fit[,X], col=pal_col[X]))
```

Figure 11.37 shows the probability of a subject giving an answer within a given 10% band, given their answer to question q31 (the formula:  $q42 \sim q5+q29+q31$ , is a better fit, but is not so easy to plot in 2-D).

Perhaps use CART when there are several categorical variables, early splits involving these variables may suggest that multiple models be created...

### 11.3.6 Rates and proportions response variables

When dealing with a response variable that is a rate or proportion, there is a fixed lower and upper bound, e.g., 0 and 100. Measurements within a fixed interval often share two characteristics: they exhibit more variation around the mean and less variation towards the lower and upper bounds,<sup>xviii</sup>, and they have an asymmetrical distribution. These characteristics can be modeled by a Beta equation. A regression model where the response variable is fitted to a Beta equation is known as a *Beta regression model*.

The `betareg` package contains functions that support the fitting of Beta regression models. When fitting basic models, calls to `betareg` have the same form as calls to `glm`; both functions include options that are not supported by the other.

Figure 11.38 shows fitted curves from a beta regression model (red), bootstrapped confidence intervals (blue) and a call to `glm` (green); the study that produced the data is discussed elsewhere, see Figure 6.45. The equation fitted is:  $\text{mutants killed} \propto \sqrt{\text{coverage}}$ , and was chosen because it is something simple that works reasonably well. Searching for the best fitting exponent, using `nls` (the `betareg` package does not support fitting non-linear models), shows that 0.44 is a better fit than 0.5 for this sample.

The summary output for a Beta regression model includes extra information, as follows: [code](#)

<sup>xviii</sup> The measurement sample is heteroskedastic.

```

Call:
betareg(formula = y_measure ~ I(x_measure^0.5))

Standardized weighted residuals 2:
    Min     1Q   Median     3Q    Max 
-2.6881 -0.6403 -0.1279  0.6399  3.1829 

Coefficients (mean model with logit link):
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -2.5460     0.1891 -13.46 <2e-16 ***
I(x_measure^0.5) 4.7093     0.3502  13.45 <2e-16 ***

Phi coefficients (precision model with identity link):
            Estimate Std. Error z value Pr(>|z|)    
(phi)      4.9641     0.5386  9.217 <2e-16 ***

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Type of estimator: ML (maximum likelihood)
Log-likelihood: 80.37 on 3 Df
Pseudo R-squared: 0.6323
Number of iterations: 13 (BFGS) + 1 (Fisher scoring)

```

The phi coefficient (the Greek letter  $\phi$ ) is the second coefficient of the fitted Beta distribution,  $B(\mu, \phi)$ .

The `predict` function returns the expected value of the response variable,  $E(y) = \mu$ , but does not support a `se.fit` option (when passed a Beta regression model). The bootstrap can be used to calculate a confidence interval from the predictions made by many models, as follows:

```

library("betareg")
library("boot")

boot_reg=function(data, indices)
{
  cov_data=data[indices, ]
  b_mod=betareg(mu_cov ~ I(path_cov^0.5), data=cov_data)
  # A vector must be returned, i.e., no data frames
  return(predict(b_mod, newdata=data.frame(path_cov=x_vals)))
}

cov_boot=boot(pm_info, boot_reg, R = 4999)

ci=apply(cov_boot$t, 2, function(X) quantile(X, c(0.025, 0.975)))
lines(x_vals, ci[1, ], col=pal_col[3])
lines(x_vals, ci[2, ], col=pal_col[3])

```

The default link function used by the `betareg` function is `logit`, the same default link used by `glm`, for the argument `family=binomial`.

## 11.4 Multiple explanatory variables

Linear regression can be used to fit models containing more than one explanatory variable; *multiple regression* is the term used for modeling with more than one explanatory variable (the term *bivariate regression* is sometimes applied to the single explanatory variable+response variable case). In theory there is no limit on the number of explanatory variables, but in practice available processing resources and the need to hold data in storage set an upper bound.

Visualization is much more complicated when there are multiple explanatory variables. Chapter 8 contains examples for visualizing two variables, and the general approach is to break down multiple regression visualization into pairs of variables.

System performance is affected by many factors, Figure 11.39 shows SPECint 2006 results for processors running at various frequencies (upper), color coded by memory chip frequency (center) and name of processor family (lower).

The SPECint results include 36 columns of information relating to the benchmarked system. Which of these columns contains information that can be used to succinctly model the performance of a system and what equation best describes the form of their contribution?

The R formula notation includes a symbol that denotes all columns in the data frame as explanatory variables, except the one specified as the response variable; the dot symbol is used as follows:

```
spec_mod=glm(Result ~ ., data=cint)
```

Given enough cpu power and memory, it can be more productive to start by considering all explanatory variables and remove underperforming variables, rather than starting with the explanatory variables believed to be the most important and then adding more variables.

The stepAIC function, in the MASS package, automates the process of removing underperforming explanatory variables from a fitted model, to create a model having a minimum AIC (the step function, in the base system, is a rather minimal implementation).<sup>xix</sup>

When some domain knowledge is available (e.g., performance often correlates with clock rate and is not usually affected by date of execution), experimentation of fitting models containing explanatory variables considered to be most likely to have a large impact on the response variable, can help refine the analyst's appreciation of the impact of different explanatory variables on overall performance.

For this SPEC dataset, there is so much detail recorded in the Processor column of the Spec results, that each entry is often unique; making it possible to create an almost perfect, but completely uninformative, model using just this one explanatory variable.

The following model explains 80% of the variance in the Result values:

```
spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)
```

where: Processor.MHz is the processor clock rate, mem\_rate the peak memory transfer rate and mem\_freq the frequency at which memory is clocked.

The + binary operator, in the above formula, specifies that explanatory variables are added together. The summary output shows that the equation fitted by glm is:

$$Result = -2.4 \cdot 10^1 + Processor.MHz 7.3 \cdot 10^{-3} + mem\_rate 2.5 \cdot 10^{-3} + mem\_freq 1.0 \cdot 10^{-2}$$

With a single explanatory variable, it is easy to visually compare model predictions against measured values; with multiple variables things are not so simple. One approach is to analyse the impact of each variable, on predictions, in turn.

The crPlot and crPlots functions, in the car package, produce a *component+residual* plot (also known as a *partial-residual* plot); the y-axis contains the predicted value plus the residual, the x-axis the value of the explanatory variable.

```
library("car")
spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)
crPlots(spec_mod, term= ~ ., layout=c(3, 1), col=point_col,
        cex.lab=1.5, cex.axis=1.5, ylab="Component+Residuals\n", main="")
```

Figure 11.40 shows the component+residual plots produced by the above code. The red dotted line is derived from the fitted model and the green line a loess fit; if the form of an explanatory variable, in the formula used to fit a model, is close to reality, the two lines will be closely intertwined. For the SPEC model there is consistent divergence of the two lines, over some of the measurement interval, for two variables and perhaps some for a third.

Experience of hardware characteristics suggests that performance does not increase forever, as clock rates are increased. Adding quadratic forms of the explanatory variables to the model

<sup>xix</sup> This fishing expedition approach to model building requires that p-values be suitably reduced, e.g., using a Bonferroni corrected value.

is a step up in complexity, to try out with a fitted model (an exponential is more realistic, in that its maximum converges to a limit, but this form of modeling requires the use of non-linear regression, which is covered later).

Adding quadratic terms, for two of the three explanatory variables, to the fitted model explains another 4% of the variance, but significantly reduces the error at higher processor and memory frequencies; see Figure 11.41.

Some of the systems benchmarked contained error correcting memory, which might be expected to slightly reduce performance. The update function can be used to add, or remove, explanatory variables from a previously fitted model. The following code adds the variable `ecc` to the previously fitted model, `spec_mod`:

```
ecc_spec_mod=update(spec_mod, . ~ . + ecc)
```

The advantage of using `update` is a reduction in the system resources needed to fit the model, compared with starting from the beginning again.

The `summary` output shows that systems containing error correcting memory have slightly better performance. Before jumping to the conclusion that adding error correction improves system performance, it is worth noting that this kind of memory tends to be used in high-end systems, where it is likely that money has been spent to improve performance and reliability.

The choice of `cpu` and memory frequency is based on information that is not present in the SPEC result data, the intended price point the computing system is designed to be sold at and the trade-off in the cost/performance of the components needed to build it.

What contribution does each explanatory variable make to a fitted model? Some of the ways in which individual contributions can be measured include:

- the amount of variance, in the response variable, explained by an explanatory variable. The `calc.relimp` function, in the `relaimpo` package, calculates the contribution made by each explanatory variable to the variance explained by the fitted model,
- the impact each explanatory variable can have on the range of values taken by the response variable (with all other explanatory variables maintaining a fixed value).

For very simple models,<sup>xx</sup> one way of calculating the maximum impact on the value of the response variable is by multiplying the minimum/maximum value taken by an explanatory variable by the corresponding coefficient in the fitted model. For instance, `range(cint$Processor.MHz)*7.3*10^-3` evaluates to 11.68 35.04, a difference of 23.36.

The `visreg` function, in the `visreg` package, produces a visual representation of the impact of each explanatory variable on the response variable.

Nomograms are a visual method for calculating the value of a response variable when each explanatory variable has a particular value: the `DynNom` function in the `DynNom` package supports interactive exploration of model behavior in a web browser.

Normalising values prior to fitting a model is sometimes suggested (e.g., using the `scale` function); the relative values of the model coefficients can then be directly compared. This method only works when all explanatory variable values are drawn from a Normal distribution.

The `relaimpo` package supports a variety of functions<sup>630</sup> for calculating the relative contribution made to a model by each explanatory variable it contains. For instance, the `calc.relimp` function calculates: `first`, the variance explained by a model containing just each variable, `last`, the variance explained when a variable is added to a model that already contains the other variables, `betasq`, the standardized coefficients of the model (i.e., one fitted after normalising the data; effectively a metric for the contribution of each explanatory variable to the value of the response variable), and `lmg` (named after the initials of its creators), the variance explained by each variable; the `boot.relimp` function returns confidence intervals for these values.

```
library("relaimpo")
spec_mod=glm(Result ~ Processor.MHz+I(Processor.MHz^2)+mem_rate
```

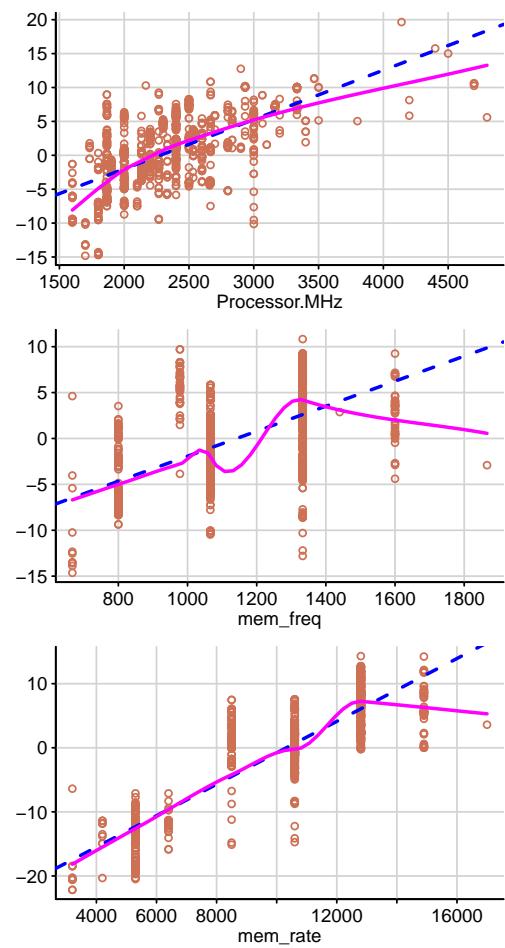


Figure 11.40: Component+residual plots for three explanatory variables in a fitted SPECint model. [code](#)

<sup>xx</sup> Those that are linear in the explanatory variable, with no interactions between variables.

```
+ I(mem_rate^2)+mem_freq, data=cint)

# How much does each explanatory variable contribute?
calc.relimp(spec_mod, type = c("first", "last", "betasq", "lmg"))
```

The calc.relimp output is: code

Response variable: Result  
 Total response variance: 81.56  
 Analysis based on 1346 observations

5 Regressors:  
 Processor.MHz I(Processor.MHz^2) mem\_rate I(mem\_rate^2) mem\_freq  
 Proportion of variance explained by model: 83.77%  
 Metrics are not normalized (rela=FALSE).

Relative importance metrics:

	lmg	last	first	betasq
Processor.MHz	0.06189	0.017807	0.04609	0.6888
I(Processor.MHz^2)	0.04556	0.005698	0.02962	0.2201
mem_rate	0.29380	0.028909	0.55554	2.0853
I(mem_rate^2)	0.29050	0.006363	0.58253	0.4768
mem_freq	0.14598	0.067531	0.28997	0.1258

Average coefficients for different model sizes:

	1X	2Xs	3Xs	4Xs	5Xs
Processor.MHz	4.308e-03	1.290e-02	1.568e-02	1.823e-02	1.666e-02
I(Processor.MHz^2)	6.560e-07	-4.894e-07	-9.880e-07	-1.728e-06	-1.788e-06
mem_rate	2.343e-03	1.562e-03	2.236e-03	3.303e-03	4.540e-03
I(mem_rate^2)	1.280e-07	1.488e-07	8.108e-08	-1.286e-08	-1.158e-07
mem_freq	2.429e-02	2.012e-02	1.558e-02	1.457e-02	1.600e-02

the second set of columns, under the line starting Average coefficients, lists the model coefficients for each explanatory variable, if that variable were to appear in a model containing X variables (values are averaged over all combinations of other variables). The values in the last column (5Xs in this case) are the same as those produced by the summary function for the fitted model.

How do changes in the value of each explanatory variable individually affect the value of the response variable? The visreg package supports functions for plotting the relationship between the response variable and individual explanatory variables (with the other variables held constant at their median value).

Figure 11.41 shows the individual contribution made by each explanatory variable to the value of the response variable (along with confidence intervals in grey), for the following model of SPECint performance:

```
library("visreg")
spec_mod=glm(Result ~ Processor.MHz + I(Processor.MHz^2)+mem_freq
            +mem_rate+I(mem_rate^2), data=cint)
visreg(spec_mod)
```

The visreg2d function creates contour and perspective plots, see Figure 11.42.

### 11.4.1 Interaction between variables

In the models fitted so far, each explanatory variable has been independent of the others. The `glm` function and many other regression modeling functions provide mechanisms for specifying interactions between explanatory variables, using binary operators in the formula, such as `:`, `*` and `^`.

As with all data analysis, the choice of interactions between explanatory variables should be driven by domain knowledge. When there is a lot of uncertainty about which interactions

Figure 11.41: Individual contribution of each explanatory variable to the response variable in a quadratic model of SPECint performance. code

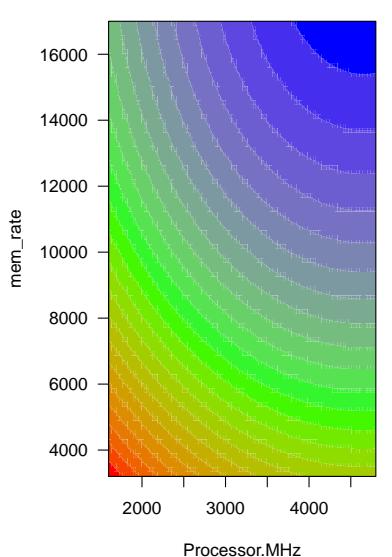


Figure 11.42: Contour map of Result values predicted by fitted model of SPECint performance, over range of Processor.MHz and mem\_rate values. code

Operator	Effect
+	causes both of its operands to be included in the equation.
:	denotes an interaction between its operands, e.g., $a:b$ or $a:b:c$ .
*	denotes all possible combinations of + and : operators, e.g., $a*b$ is equivalent to $a+b+a:b$ .
$\wedge$	denotes all interactions to a specific degree, e.g., $(a+b+c) \wedge 2$ is equivalent to $a+b+c+a:b+a:c+b:c$ .
.	denotes all variables in the data-frame specified in the data argument except the response variable.
-	specifies that the right operand is removed from the equation, e.g., $a*b-a$ is equivalent to $b+a:b$ .
-1	specifies that an intercept is not to be fitted (many regression fitting functions implicitly include an intercept).
I0	"as-is", any operators in the enclosed expression are not treated as formula operators, their behavior that which occurs outside of a formula.

Table 11.2: Symbols that can be used within a formula to express relationships between explanatory variables.

are significant, it may be easiest to start by specifying all pairs of interactions between variables (or triple interactions if there are not too many variables), and to then simplify, either automatically using `stepAIC` or through manual inspection of `summary` output of the fitted models.

Stepwise regression techniques, such as that provided by `stepAIC`, can return models that suffer from a variety of problems, such as overfitting. There are techniques available to help avoid these problems; the `train` function in the `caret` package supports some of these techniques. The `glmulti` package automates the process of finding an optimal, in a sense specified by the user (e.g., minimise AIC or some other measure), explanatory variable interaction; a list of variables is specified and the function permutes through the possibilities, e.g., `glmulti("y", c("a", "b", "c", "d"), data=some_data)`.

A study by Moløkken-Østvold and Furulund<sup>1103</sup> investigated the impact of daily communication between customer and contractor on the accuracy of effort estimates, for 18 software projects. Figure 11.43 shows estimated vs. actual effort broken down by communication frequency (i.e., daily or not daily), along with individually fitted straight lines.

It is possible to fit one regression model that simultaneously fits both straight lines to this data; the following code shows one possibility:

```
sim_mod=glm(Actual ~ Estimated+Estimated:Communication, data=sim)
```

The fitted equation is (based on `summary` output):

$$\begin{aligned} \text{Actual} &= -270.1 + 1.18\text{Estimated} + 0.51\text{Estimated} \times D \\ &= -270.1 + (1.18 + 0.51D) \times \text{Estimated} \end{aligned}$$

where: *Actual* is the actual and *Estimated* the estimated effort, and *D* has one of two values:

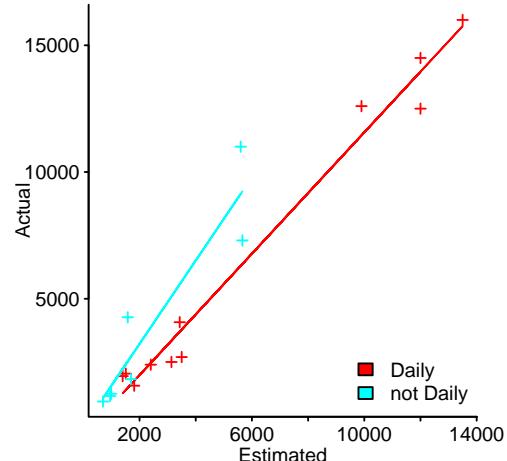
$$D = \begin{cases} 1 & \text{daily communication} \\ 0 & \text{not daily communication} \end{cases}$$

Is this formula the best fit possible using the available data? The formula used was selected by your author, because of a belief that the benefit of communication will increase as project size increases.

There are six data points for each of the 18 projects, computationally small enough for the brute force approach of examining all possible models; but with only 18 projects, some formula possibilities cannot be fitted because they contain more variables than available data points (a unique solution requires fewer variables than data points).

The formula in the following code fits four explanatory variables individually, plus each variable paired with every other variable (one at a time). `stepAIC` is used as a quick way of removing explanatory variables that are not paying their way (automatic model selection is fraught with problems, with perhaps the largest being that it allows analysts to stop thinking about the data):

```
sim_mod=glm(Actual ~ (Estimated+Communication+Contract+Complexity)^2, data=sim)
```

Figure 11.43: Estimated and actual effort broken down by communication frequency, along with individually fitted straight lines. Data from Moløkken-Østvold et al.<sup>1103</sup> code

```
min_sim=stepAIC(sim_mod)
summary(min_sim)
```

This book's primary aim is to fit models, as a means of building understanding, and minimising AIC is often a useful step along the way. Another way of removing low impact variables from a model is to consider the p-value of each fitted component.

The `summary` output for ordinal and nominal explanatory variables, lists p-values for each value that these variables take in the data. The `Anova` function (in the `car` package) lists p-values at the variable level, and its output for the above model is: `code`

Analysis of Deviance Table (Type II tests)

	Response: Actual	LR	Chisq	Df	Pr(>Chisq)						
Estimated		45.879	1	1.258e-11	***						
Communication		17.272	1	3.240e-05	***						
Contract		6.767	3	0.07971	.						
Complexity		1.543	1	0.21423							
Estimated:Communication		2.546	1	0.11060							
Communication:Contract		5.020	3	0.17034							
Contract:Complexity		3.197	2	0.20224							
---											
Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'. '	0.1	' '	1

The variable `Complexity` has the highest p-value, and repeatedly removing the component having the highest p-value, for successively smaller models (smaller in the sense of containing fewer components) leads to the following model:

```
sim_mod=glm(Actual ~ Estimated+Communication+Communication:Contract, data=sim)
```

which corresponds to the equation:

$$\begin{aligned} \text{Actual} = & -274.8 + 1.21\text{Estimated} + 2625 \times !D + \\ & C_{fp}(1862 \times !D - 197.6 \times D) + \\ & C_{tp}(-2270 \times !D - 462.2 \times D) + \\ & C_{ot}(-2298 \times !D - 234.3 \times D) \end{aligned}$$

where the new variables are:  $C_{fp}$  is a fixed price contract,  $C_{tp}$  is a target price contract and  $C_{ot}$  other kind of contract.

$$C_{fp} = \begin{cases} 1 & \text{fixed price contract} \\ 0 & \text{not fixed price contract} \end{cases} \quad C_{tp} = \begin{cases} 1 & \text{target price contract} \\ 0 & \text{not target price contract} \end{cases} \quad C_{ot} = \begin{cases} 1 & \text{other contract} \\ 0 & \text{not other contract} \end{cases}$$

This model explains a greater percentage of the variance in the data than the first model fitted, it also has a slightly smaller AIC. While it makes use of extra information (i.e., the kind of contract), a more noticeable difference is that `Communication` has a constant effect (i.e., it does not increase with estimated size); the case of fixed price contracts, with no daily communication cries out for attention.

Following the numbers has produced a model which is a better fit to the data, but not to expectations (which may, of course, be wrong).

### 11.4.2 Correlated explanatory variables

The mathematics behind many of the approaches used to fit linear regression models assumes that explanatory variables are independent of each other. If a linear relationship exists between one or more pairs of explanatory variables (i.e., a relationship of the form:  $PV_1 = a + b \times PV_2$ , where  $PV_1$  and  $PV_2$  are explanatory variables,  $a$  is any constant and  $b$  a non-zero constant), then this needs to be taken into account by the model building technique used.<sup>xxi</sup>

*Multicollinearity* is said to occur, when a linear relationship exists between two or more explanatory variables, the term *collinearity* is often used when only two variables are involved.

<sup>xxi</sup> It is ok for a nonlinear relationship to exist in linear models, e.g.,  $PV_1 = a + b \times PV_2^2$ .

Figure 11.44 illustrates how the variance in  $Y$  explained by combining  $X_1$  and  $X_2$  may be less than the sum of the variance explained by each individually, because the two variables are not independent; there is a shared contribution.

The impact of multicollinearity is to increase the standard error in the calculated value of the coefficients of the fitted model (i.e., the  $\beta_n$ ), potentially resulting in a model that is not considered acceptable or is unreliable (in the sense that small changes in the data result in large changes in the coefficients of the fitted model). The increased uncertainty, in some variables, will make it more difficult to isolate the effects of individual explanatory variables and will increase the width of the confidence intervals for the predicted values of the response variable.

The *Variance Inflation Factor* (VIF) is a measure of the uncertainty created by the presence of multicollinearity. The impact of VIF is the same as reducing the sample size. When no multicollinearity is present, VIF has a value of one. The impact on the standard error is:

$$\epsilon_{\text{standard}} \propto \sqrt{\frac{\text{VIF}}{\text{observations}}}$$

When is a VIF value too large? A large VIF is more likely to be acceptable with a large sample, compared to a small one, e.g., the standard error is proportionally the same for 10,000 observations having a VIF of 400 and for 100 observations having a VIF of 4.

Suggested maximum VIF values appear in print, e.g., 5 or 10 are sometimes suggested. As always, think about what the VIF value means in the context of how the results will be used; pick a value that makes sense given the sample size, the error in the measurements and the level of error that is acceptable in the business context.

The car and rms packages support a vif function, that takes the model returned by a call to, for instance, `glm` and returns the VIF for each explanatory variable.

A study by Kroah-Hartman<sup>885</sup> investigated the amount of change in the Linux kernel source code occurring between each release. Figure 11.45 shows the number of lines added, modified and removed, plus overall growth, number of files and total number of lines at each initial release of the Linux kernel from version 2.6.0 to 3.9 (two outliers have been excluded).

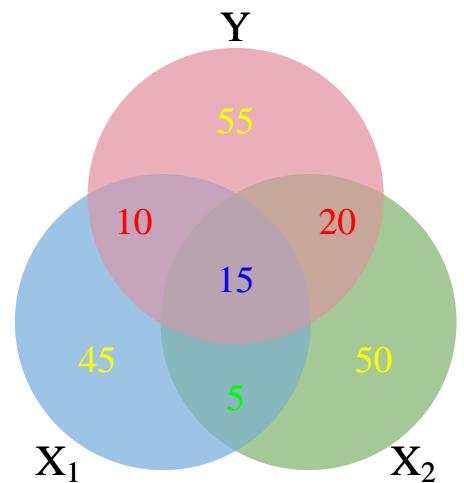
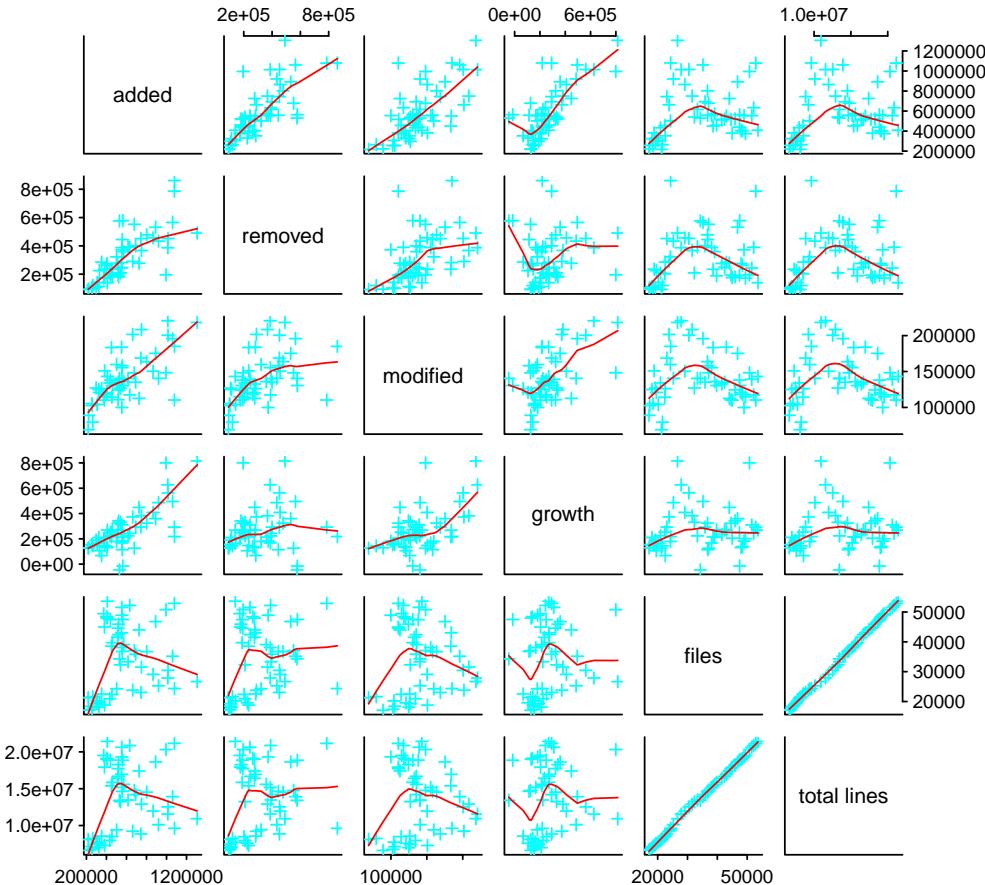


Figure 11.44: Illustration of the shared and non-shared contributions made by two explanatory variables to the response variable  $Y$ . [code](#)



Building a model of the growth of the Linux kernel is complicated by the potentially high correlation between some of the measured variables, including:

Figure 11.45: pairs plot of lines added/modified/removed, growth and number of files and total lines in versions 2.6.0 through 3.9 of the Linux kernel. Data from Kroah-Hartman.<sup>885</sup> [code](#)

- the growth, in lines of code, between releases is the difference between lines added and lines removed; these three variables are perfectly correlated in that knowing two of them enables the third to be calculated,
- lines added appears strongly correlated with lines removed. Perhaps existing functionality is being rewritten, rather than unrelated functionality being added,
- the decision about whether a line has been modified or removed/added is made algorithmically (rather than asking the developer who made the change). The amount of misclassified lines is not known,
- system level measurements are also correlated, e.g., number of files and total lines of code.

Modeling the number of modified lines, using the Kroah-Hartman data, finds that both lines added and lines removed individually explain around half of the deviance (61% and 41% respectively). However, combining them in a model does not produce any improvement; the following output from `summary` was obtained by including the argument `correlation=TRUE`. `code`

Call:

```
glm(formula = lines.modified ~ lines.added + lines.removed, data = amr_out)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-72376	-12049	321	11274	54964

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	8.705e+04	8.625e+03	10.093	9.40e-14 ***
lines.added	9.958e-02	2.093e-02	4.759	1.64e-05 ***
lines.removed	-1.500e-02	3.117e-02	-0.481	0.632

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 639477748)

Null deviance: 6.2276e+10 on 53 degrees of freedom  
 Residual deviance: 3.2613e+10 on 51 degrees of freedom  
 AIC: 1253.1

Number of Fisher Scoring iterations: 2

Correlation of Coefficients:

	(Intercept)	lines.added
lines.added	-0.54	
lines.removed	-0.06	-0.76

The correlation between the model coefficients appears at the end of the output and shows a high negative correlation between `lines.added` and `lines.removed`; the variable `lines.added` is a better predictor of `lines.modified` and has been selected over `lines.removed` (whose p-value is significantly larger than when just this variable appeared in a model).

A call to the `vif` produces the following: `code`

```
lines.added lines.removed
2.39311      2.39311
```

With only two explanatory variables, there is no ambiguity about which variables are involved in a linear relationship, but with more than two variables things are not always so obvious. The correlation table produced by `summary` can be used to identify related variables; the `alias` function generates just this information, when the argument `partial=TRUE` is specified.

Approaches to dealing with multicollinearity, to reduce any undesirable impact it may have on fitting a model, include:

- removing one or more of the correlated explanatory variables. The choice of which explanatory variables to remove might be driven by:

- the cost of collecting information on the variable(s),
- a VIF driven approach. The process involves fitting a model using the current set of explanatory variables, removing the explanatory variable with the largest VIF (removing one variable affects the VIF of those that remain and may reduce the VIF of other variables to an acceptable level) and iterating until all explanatory variables have what is considered to be an acceptable VIF,
- combining the strongly correlated variables in a way that makes use of all the information they contain.

The disadvantages of excluding explanatory variables from a model include:

- ignoring potentially useful information present in the excluded variable,
- creating a model that gives a false impression about which explanatory variables are important, i.e., readers will assume that the variables appearing in the model are the only important ones, unless information about the excluded variables is also provided,
- it provides the opportunity for the analyst to select the model that favours the hypothesis they want to promote (by selecting which explanatory variables appear in the model).

The SPEC power benchmark<sup>1472</sup> is designed to measure single and multi-node server power consumption, while executing a known load. The results contain 515 measurements of six system hardware characteristics, such as number of chips, number of cores and total memory, as well as average power consumption at various load factors.

A model of average power consumption, at 100% load, containing a linear combination of all explanatory variables, shows very high multicollinearity for the number of chips (its VIF is 27.5 and several other variables have a high VIF; see [hardware/SPECpower.R](#)). Removing this variable reduces the VIF of the remaining variables, but the AIC drops from 6798.7 to 7182.1. Whether this decrease in model performance is important depends on the reason for building the model, e.g., prediction or understanding. Do the values of the model coefficients, after removing this variable, provide more insight than the coefficient values of the original model? These kinds of questions can only be answered by a person having detailed domain knowledge. This example shows how removing a variable solves one problem and raises others.

A study of fault prediction by Nagappan, Zeller, Zimmermann, Herzig and Murphy<sup>1129</sup> produced data containing six explanatory variables having an exact linear relationship with other explanatory variables. The `g1m` function detects the existence of this relationship and makes a decision about explanatory variables to exclude from the model (the value returned for their fitted coefficients is NA); see [regression/change-burst-sum.R](#).

Two explanatory variables having an exact linear relationship will have a correlation of  $\pm 1$ , as a call to `alias` will show.

### 11.4.3 Penalized regression

*Penalized regression* handles multicollinearity by automatically selecting how much each explanatory variable should contribute to the model; explanatory variables are penalized, based on their relative contribution to the model.

The traditional technique for fitting a regression model involves minimising some measure of the error, where the error is defined to be the difference between actual and predicted values, e.g., the sum of squared error, whose equation is:

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Penalised regression modifies this equation to include a penalty (the  $\lambda$  in the equation below), for the  $P$  coefficients in the model ( $\beta$  in the equation below).

$$SSE_{enet} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^P |\beta_j| + \lambda_2 \sum_{j=1}^P \beta_j^2$$

This technique of using both first- and second-order penalties is known as the *elastic net*. When only the first order term,  $\lambda_1$ , is used, it is known as the Least Absolute Shrinkage and Selection Operator method (*lasso*). When only the second order term,  $\lambda_2$ , is used, it is known as *ridge regression*.

In theory the penalization penalties,  $\lambda_1$  and  $\lambda_2$ , are chosen by the analyst. In practice, software packages provide a function that automatically finds values (using the bootstrap) that minimise the error.

The lasso tends to pick one from each set of correlated variables and ignores the rest (by setting the corresponding  $\beta$ s to zero). Ridge regression has the effect of causing the coefficients,  $\beta$ , of the corresponding correlated variables to converge to a common value, i.e., the coefficient chosen for  $k$  perfectly correlated variables is  $\frac{1}{k}$ th the size chosen, had just one of them been used.

The calculation of mean squared error adds contributions from both variance and bias. The default regression modeling techniques are unbiased (i.e., attempt to minimise bias). It is possible to build models with lower MSE by trading off bias for variance; one consequence of correlation between variables is that the variance is high...

The penalized package...

now if we had some data where it makes a big difference (see [hardware/SPECpower.R](#) for a case where this makes a small difference)...

## 11.5 Non-linear regression

The regression models fitted in earlier sections are linear models because the coefficients of the model (e.g.,  $\beta_1$  in the equation at the start of this chapter) are linear (the form of the explanatory variables is irrelevant). In a non-linear regression model one or more of the coefficients have a non-linear form, e.g.,  $\beta_2$  in the following equation:

$$y = \alpha + \beta_1 x^{\beta_2} + \epsilon$$

Table 11.3 lists some commonly occurring non-linear equations and Figure 11.46 illustrates example instances of these equations.

The `nls` function (Nonlinear Least Squares) is part of the base system and can be used to build non-linear regression models; it requires that the response variable error have a Normal distribution (`glm`'s default behavior). The `gnm` package (Generalized nonlinear models) contains support for other forms of error distribution.

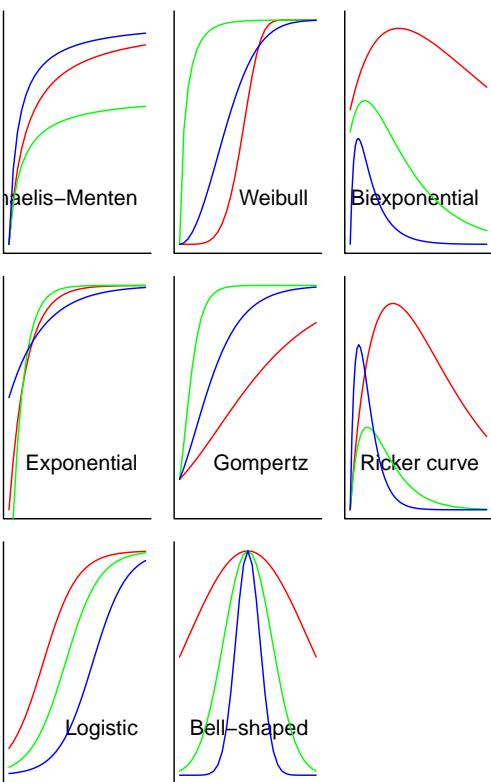


Figure 11.46: Example plots of functions listed in Table 11.3. These equations can be inverted, so they start high and go down. [code](#)

Shape	Name	Equation
Asymptotic growth to a limit	Michaelis-Menten	$y = \frac{ax}{1+bx}$
Asymptotic growth to a limit	Exponential	$y = a(1 - be^{-bx})$
S-Shaped	Logistic	$y = a + \frac{b-a}{1+e^{(c-x)/d}}$
S-Shaped	Weibull	$y = a - be^{-cx^d}$
S-Shaped	Gompertz	$y = ae^{be^{-cx}}$
Humped	Bell-shaped	$y = ae^{- bx ^2}$
Humped	Biexponential	$y = ae^{-bx} - ce^{-dx}$
Humped	Ricker curve	$y = axe^{-bx}$

Table 11.3: Some commonly encountered non-linear equations, see Figure 11.46.

From the practical point of view there are several big differences between using `glm` and using `nls`, including:

- `nls` may fail to fit a model; the techniques used to find the coefficients of a non-linear model are not guaranteed to converge,
- `nls` may return a fitted model that differs from the actual solution; the techniques used to find the coefficients of a non-linear model may become stuck in a local minimum, that is good enough, and fail to find a better solution,

- nls often requires the analyst to provide estimate(s) for the initial value of each model coefficient, that is close to the final values (using the start argument),
- names for the model coefficients being estimated have to explicitly appear in the formula (i.e., implicit names are not created automatically),
- the operators appearing in the expression to the right of ~ have their usual arithmetic interpretation, i.e., the formula specific behaviors listed in Table 11.2 do not apply.

The biggest problem with fitting non-linear regression models, is finding a combination of starting values that are good enough for nls be able to converge to a fitted model. Possible techniques for finding these values include:

- using a "self-start" function, if available (e.g., SSlogis for Logistic models); these attempt to find good starting values to feed into nls, and functions, in turn, may require starting values (but at least there is a known method for calculating them),
- fitting a linear model that is close enough to the non-linear model and working with the coefficients of the fitted linear model as possible starting values,
- using the argument trace=TRUE, which outputs the list of model coefficients that are being used internally, as a source of ideas,
- picking a few points in the plotted data that a fitted line is likely to pass through and calculating values that would result in the equation being fitted, passing close to these points.

A study by Hazelhurst<sup>671</sup> measured the performance of various systems running a computational biology program. Figure 11.47 shows four non-linear equations fitted to one processor characteristic (L2 cache size). The calls to nls are as follows:<sup>xxii</sup>

```
b_mod=nls(T1 ~ c+a*exp(b*L2), data=bench, start=list(a=300, b=-0.1, c=60))
mm_mod=nls(T1 ~ (1+b*L2)/(a*L2), data=bench, start=list(b=3, a=0.004))
gm_mod=nls(T1 ~ a/exp(b*exp(-c*L2)), data=bench,
           start=list(a=80, b=-1, c=0.1), trace=FALSE)
Asym = 0.0125
Drop = 0.002
lrc = -1.0
pwr = 2.5
# 1/SSweibull does not have the desired effect, so have to invert the response.
getInitial(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
wb_mod=nls(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
```

At the start of this chapter, various linear models were fitted to the growth of Linux, see Figure 11.7. Polynomials containing integer powers were used, perhaps the data is better fitted by a polynomial containing non-integer powers. The following call to nls attempts to fit such an equation, it uses starting values extracted from the quadratic model fitted earlier:

```
m1=nls(LOC ~ a+b*Number_days+Number_days^c, data=h2,
       start=list(a=3e+05, b=-4e+2, c=2.0))
```

The summary and AIC output is: [code](#)

Formula: LOC ~ a + b \* Number\_days + Number\_days^c

Parameters:

	Estimate	Std. Error	t value	Pr(> t )
a	-1.679e+05	2.969e+04	-5.656	2.61e-08 ***
b	7.319e+02	3.463e+01	21.131	< 2e-16 ***
c	1.806e+00	4.616e-03	391.211	< 2e-16 ***

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 231800 on 498 degrees of freedom

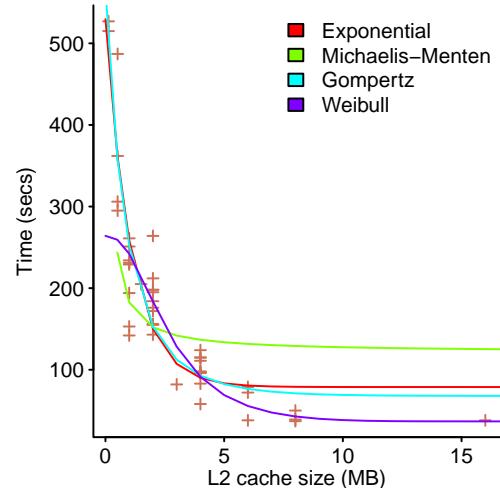


Figure 11.47: Time to execute a computational biology program on systems containing processors with various L2 cache sizes. Data kindly provided by Hazelhurst.<sup>671</sup>  
[code](#)

<sup>xxii</sup> It is difficult to separate inspiration from suck it and see, in this process.

Number of iterations to convergence: 5  
Achieved convergence tolerance: 4.299e-06

[1] "AIC = 13805.2100165816"

showing that the equation:

$$sloc = (-1.68 \cdot 10^5 \pm 3 \cdot 10^4) + (7.32 \cdot 10^2 \pm 3.5 \cdot 10^1) Number\_days + Number\_days^{1.81 \pm 4.6 \cdot 10^{-3}}$$

is a slightly better fit than a cubic equation (i.e., a lower AIC) and also predicts continuing growth (unlike the cubic equation).

It is possible that further experimentation will find a polynomial model with a lower AIC. However, the purpose of this analysis is to understand what is going on, not to find the equation whose fitted model has the lowest AIC.

A more practical issue is to create a model that makes what are considered to be more realistic future predictions. The growth in the number of lines in the Linux kernel will not continue forever, at some point the number of lines added will closely match the number of lines deleted. One commonly seen growth pattern, starts slow, has a rapid growth period, followed by a levelling off converging to an upper limit (i.e., an S-shaped curve). The Logistic equation is S-shaped and is often used to model this pattern of growth; the equation involves four unknowns (third row in Table 11.3).

Fitting a Logistic equation the hard and easy (when it works) way:

```
# suck it and see...
m3=nls(LOC ~ a+(b-a)/(1+exp((c-Number_days)/d)), data=h2,
      start=list(a=-3e+05, b=4e+06, c=2000, d=800))
# no thinking needed, SSfpl works out of the box for this data :-)
m3=nls(LOC ~ SSfpl(Number_days, a, b, c, d), data=h2)
```

The AIC for the fitted Logistic equation is slightly worse than the cubic polynomial (13,273 vs. 13,220), but a lot better than the quadratic fit and it predicts a future trend that is likely to occur, eventually.

While the predict function includes parameters to request confidence interval and standard error information, support for both is currently unimplemented for models fitted using nls. The confint function in the MASS package, when passed a model built using nls, returns the confidence intervals for each of the model coefficients; bootstrapping can also be used to find confidence intervals.

Figure 11.48 shows the fitted model predicting a slow down in growth, with the maximum being reached at around 10,000 days. Who is to say whether this prediction is more likely to occur, over the specified number of days, than the continuing increase predicted by the quadratic model? Given that the one explanatory variable used to fit the models, time, does not directly impact the production of source, it is no surprise that the predictions of future behavior made by the various models vary so wildly.

One technique for getting a rough idea of the accuracy of the future predictions made by a model, is to fit models to subranges of the data, and then check the predictions made against the known data outside the subrange. Figure 11.49 shows logistic equations fitted to subranges of the data, e.g., all data up to 2900, 3650, 4200 number of days and all days.

The lesson to learn from Figure 11.49 is to be careful what you ask for, asking for a logistic equation fitted to the data, may get you one. The fitting process is driven by your expectations (in the form of a formula) and the data it is given.

The processes generating the data fitted by a Logistic equation may not in themselves follow this pattern, the contributions of independent processes may combine to create an emergent pattern. A study by Grochowski and Fontana<sup>629</sup> showed that increases in the density of data stored on hard disks could be viewed as a sequence of technologies that each rapidly improved (e.g., magneto-resistive and antiferromagnetically-coupled). Figure 11.50 shows the areal density (think magnetic domains) of various models of hard disk on first entering production. Improvements in each technology can be fitted with its own Logistic equation, as can the overall pattern of performance improvements.

A codebase showing some evidence of having completed its major expansion phase is glibc (i.e., its growth rate has levelled off), the GNU C library; see Figure 11.51. Plugging the fitted model coefficients into the Logistic equation we get:

$$LOC = -28168 + \frac{1114626 + 28168}{1 + e^{(3652 - Days)/935}}$$

Since these measurements were made, the C Standard's committee, JTC1 SC22/WG14, have started work on revising the existing specification; so the model's prediction that glibc will max out at around 1,115,000 lines is unlikely to remain true for many more years.

A study by Chen, Groce, Fern, Zhang, Wong, Eide and Regehr<sup>282</sup> investigated faults in a C compiler and JavaScript engine, by having them process randomly generated programs. Some programs failed to be correctly processed (1,298 in gcc and 2,603 in Mozilla's SpiderMonkey) and many of these failures could be traced back to the same few underlying faults, i.e., some faults were encountered more often than others. Figure 11.52 shows the number of failing programs that could be traced back to the same fault, the curved green line is a regression fit (a biexponential, or double exponential); the two straight lines are the exponentials that are added to form the bi-exponential.

The nls has a SSbiexp starter function, which performs poorly for this data (or, at least, your author could not make it do well).

The sample contains count data, with many very small values, implying a Poisson error distribution. The gnm function, in the gnm package, has an option to select an error distribution.

The formula notation used by gnm is based on function calls,<sup>1572</sup> rather than the binary operators used by glm and nls. The formula argument in the following call (used to fit the model plotted in Figure 11.52), contains two exponentials (specified using the instances function), the literal 1 is a placeholder for an unknown constant multiplied (the Mult function) by an exponential (the Exp function); as with calls to nls, starting values are required:

```
library("gnm")

fail_mod=gnm(count ~ instances(Mult(1, Exp(ind)), 2)-1,
             data=wrong_cnt, verbose=FALSE,
             start=c(2000.0, -0.6, 30.0, -0.1),
             family=poisson(link="identity"))
```

A possible reason for why the biexponential is such a good fit is discussed elsewhere, see Figure 6.19.

### 11.5.1 Power laws

Plotting values drawn from a power law distribution using a log scale for both axis, produces a straight line. This straight line characteristic is not unique to power laws, it can also occur with samples drawn from other distributions, e.g., an exponential distribution.<sup>xxiii</sup>

The poweRlaw package includes functions for fitting and checking whether a power law is likely to be a good fit for a sample.<sup>302</sup>

When the model being fitted contains one explanatory variable, thought to have the form of a power law, functions from the poweRlaw package can be used. However, this package does not support more complicated models, and so other regression modeling functions have to be used when a power law is one of multiple components in a model, e.g., nls.

A study by Queiroz, Passos, Valente, Hunsen, Apel and Czarnecki<sup>1298</sup> analysed the conditional compilation directives (e.g., #ifdef) used to control the optional features in 20 systems written in C. Researchers in this area use the term *feature constant* to denote macro names used to control the selection of optional features and *scattering degree* to describe the number of ifdefs that refer to a given feature constant, e.g., if the macro SUPPORT\_X appears in two ifdefs, it has a scattering degree of two.

Figure 11.53 shows the total number of feature constants (y-axis) having a given scattering degree (x-axis) in these 20 systems, lines are a power law (red) and exponential (blue) of fitted models; the numbers are the p-values for the fit (higher is better, i.e., fail to reject the hypothesis). This analysis is a fishing expedition involving 20 systems, and a power law is suggested by the visual form of the plotted data; with multiple tests it is necessary to take into account the increased likelihood of a chance match.

<sup>xxiii</sup> Papers<sup>980</sup> claiming to have found a power law, purely on the basis of a plot showing points scattered roughly along a straight line, are surprisingly common.

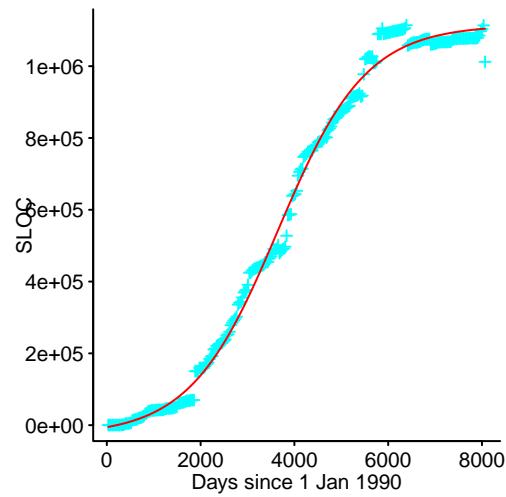


Figure 11.51: Lines of code in the GNU C library against days since 1 January 1990. Data from González-Barahona.<sup>591</sup> [code](#)

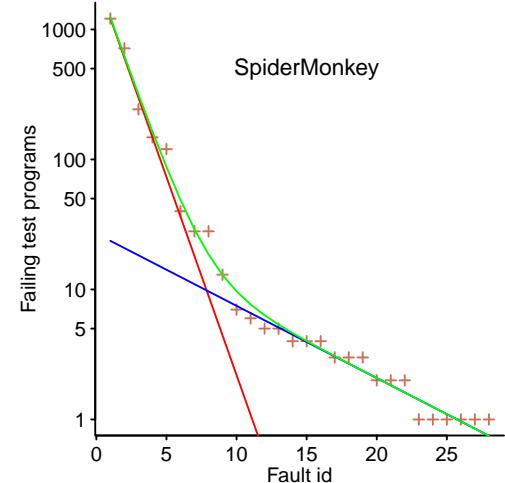
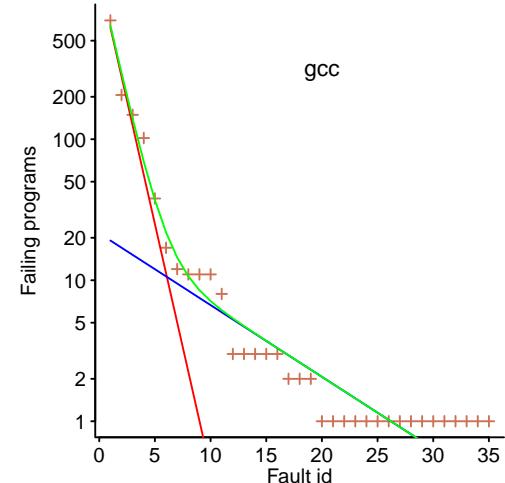


Figure 11.52: Number of failing programs caused by unique faults in gcc (upper) and SpiderMonkey (lower). Fitted model in green, with two exponential components in red and blue. Data kindly provided by Chen.<sup>282</sup> [code](#)

If 0.05 is taken as the p-value cutoff, for one test, below which the distribution hypothesis is rejected, then  $(1 - 0.95^{20}) \rightarrow 0.64$  is the cutoff when 20 tests are involved. Some systems have p-values above the cutoff for one of the power law or exponential fitted models and so the given distribution is not rejected for these systems.

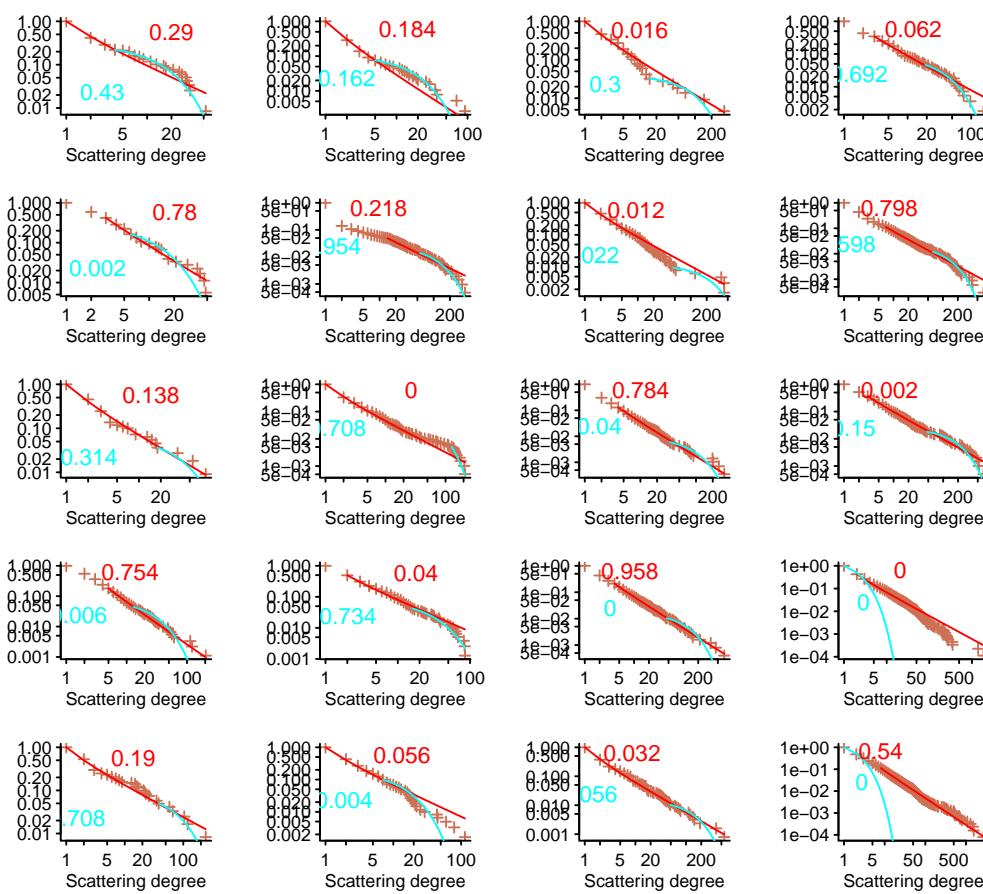


Figure 11.53: Power law (red) and exponential (blue) fits to feature macro usage in 20 systems written in C; fail to reject p-value for 20 systems is 0.64. Data from Queiroz et al.<sup>1298</sup> code

The `poweRlaw` package supports discrete and continuous forms of heavy tailed distributions. The scattering degree is an integer value and the following code fits both a discrete power law and exponential to the data (the continuous forms are `conpl` and `conexp` respectively):

```
library("poweRlaw")

# Fit scattering degree
# displ is the constructor for the discrete power law distribution
pow_mod=displ$new(FS$sd)
exp_mod=disexp$new(FS$sd) # discrete power exponential

# Estimate the lower threshold of the fit
pow_mod$setXmin(estimate_xmin(pow_mod))
exp_mod$setXmin(estimate_xmin(exp_mod))

# Plot sample values
plot(pow_mod, col=point_col, xlab="Scattering degree", ylab="")
lines(pow_mod, col=pal_col[1]) # Plot fitted line
lines(exp_mod, col=pal_col[2])

# Bootstrap to test hypothesis that sample drawn from a power law
bs_p=bootstrap_p(pow_mod, threads=4, no_of_sims=500)
text(40, 0.5, bs_p$p, pos=2, col=pal_col[1]) # Display value
```

The power law equation includes a minimum value of  $x$ , scattering degree in this case, below which it does not hold. The `estimate_xmin` function estimates the value,  $x_{min}$ , that minimises the error between the fitted model and the data. The new function, called by the constructor, sets  $x_{min}$  to the minimum value present in the data. It is common for power laws to fit a subset of the data.

## 11.6 Mixed-effects models

A study by Balaji, McCullough, Gupta and Agarwal<sup>98</sup> measured the power consumption of six different Intel Core i5-540M processors executing the SPEC2000 benchmark at various clock frequencies. The six processors measured are a sample of the entire population of Intel Core i5-540M processors. The power consumption characteristics might be modeled using the combined data from all six processors; the following is the summary output for this model: [code](#)

Call:

```
glm(formula = meanpower ~ frequency, data = power_bench)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.5746	-0.1882	0.0413	0.1902	2.2965

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	2.12594	0.01506	141.2	<2e-16 ***
frequency	1.95248	0.00767	254.6	<2e-16 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.1429928)

Null deviance: 10692.7 on 9980 degrees of freedom

Residual deviance: 1426.9 on 9979 degrees of freedom

AIC: 8916.2

Number of Fisher Scoring iterations: 2

This model does not provide any information about how performance varies between processors. The identity of the processor measured could be included in the model (see [regression/hotpower-proc.R](#)), but this is a model of the sample and it cannot be used to deduce anything about the population from which it was drawn.

How might the sample be modeled to estimate population variability? Possible techniques include:

- build a regression model for each processor, and average these six models in some way, e.g., use the coefficients from each of the six models to build a regression model that is a model of models,

Electronic circuit theory tells us that processor power consumption is proportional to clock frequency and Figure 11.54 shows the results of fitting a separate straight line to the data for each processor.

- build a *mixed-effects model*. A mixed-effects model (also known as a *hierarchical model*) might be viewed as a model of models; mathematically it uses a more direct approach, making more effective use of the available data than the method described above.

In a mixed-model the explanatory variables are classified as either a *fixed-effect* or a *random-effect* (sometimes called a *covariate*). Technically the effects are not fixed and are not random<sup>xxiv</sup>. One way to think about classifying the two kinds of explanatory variables, is to look at the impact they have on the response variable:

- fixed effects influence the mean value of the response variable and are associated with the entire population,
- random effects influence the variance of the response variable and are associated with individual subjects.

<sup>xxiv</sup> Some authors point this out and then proceed to use what they consider to be more technically correct terms, this book follows common usage because it is common; these terms crop up as named parameters in functions and appear in output information.

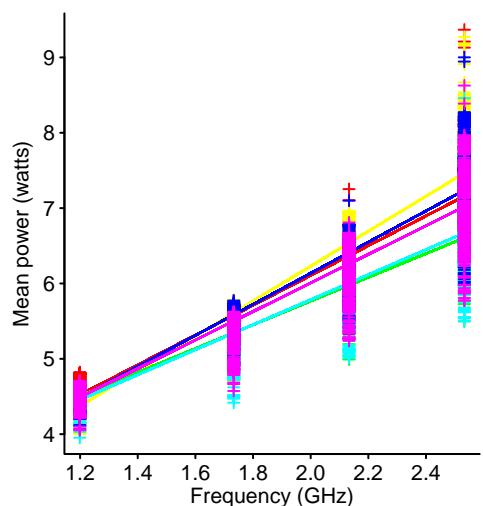


Figure 11.54: Power consumption of six different Intel Core i5-540M processors running at various frequencies; colored lines denote fitted regression models for each processor. Data from Balaji et al.<sup>98</sup> [code](#)

Mixed-effects models are used to model measurements of clusters of related subjects, and multiple correlated measurements of the same subjects (e.g., before/after measurements of the same subject).

A number of different packages are available for fitting mixed-effects models, this book uses `lme4`, whose workhorse functions are the `glmer` and `lmer` functions.<sup>xxv</sup>

The `lme4` package extends the formula notation to support the specification of random effects. In the following code:

```
library("lme4")

# Express in Gigahertz (otherwise lmer does not converge)
power_bench$frequency=power_bench$frequency/1000000

p_mod=lmer(meanpower ~ frequency + (1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency-1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency | processor), data=power_bench)
```

- first call to `lmer`: `frequency` is the fixed-effect and `(1 | processor)` is the random-effect; the `1` specifies variation in the value of the intercept and the source of this variation is the `processor` variable (i.e., the column having this name in the data frame). When plotted the model might look like the upper plot of Figure 11.55 with six lines intersecting the y-axis at different points, but all having the same slope,
- second call to `lmer`: the operand `(frequency-1 | processor)` specifies there is variation in the slope and the source of this variation is the `processor` variable. When plotted the model might look like the lines in the middle of Figure 11.55, where all lines intersect the y-axis at the same point but have different slopes.
- third call to `lmer`: the operand `(frequency | processor)` specifies that variation in the `processor` variable causes both the intercept and the slope to vary. When plotted, the models might look like the lines in the lower plot in Figure 11.55, where the lines have different intersections and slopes.

The following is the summary output from a mixed-effects model where the `processor` is a random effect on both the intercept and slope: `code`

```
Linear mixed model fit by REML ['lmerMod']
Formula: meanpower ~ frequency + (frequency | processor)
Data: power_bench
```

REML criterion at convergence: 6300.3

Scaled residuals:

Min	1Q	Median	3Q	Max
-4.0533	-0.4866	0.1453	0.4994	6.6743

Random effects:

Groups	Name	Variance	Std.Dev.	Corr
processor	(Intercept)	0.12904	0.3592	
	frequency	0.07383	0.2717	-0.99
Residual		0.10941	0.3308	

Number of obs: 9981, groups: processor, 6

Fixed effects:

	Estimate	Std. Error	t value
(Intercept)	2.1740	0.1473	14.76
frequency	1.9156	0.1111	17.23

Correlation of Fixed Effects:

(Intr)
--------

frequency -0.993
------------------

---

<sup>xxv</sup> A call to the `glmer` function that uses the default family distribution, i.e., `gaussian`, generates a warning that this usage is deprecated and `lmer` should be used.

The estimated coefficients, listed under `Fixed effects:`, for (Intercept) and frequency are very similar to the combined data model fitted earlier.

Annoyingly the summary output does not include p-values. These can be obtained using the `Anova` function from the `car` package.

The Random effects: table lists the variation introduced by processor (listed in the Groups column, on the variables listed in the Name column); Residual lists the residual left after taking all the specified random effects into account.

Taking frequency as an example, there are two sources of uncertainty in its contribution to the response variable (as expressed in its model coefficient), one from fixed effects and a random effect caused by processor variation.

Plotting the 95% confidence intervals, for the intercept and slope of a mixed-effects model, provides a visualization of the relative contribution of the sources of variation. Figure 11.56 was generated by the following code, using data from the six processors:

```
library("lattice")
library("lme4")
library("gridExtra")

proc_mod=lmer(meanpower ~ frequency +(frequency | processor),
              data=power_bench)
dp_orig=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

power_bench$shift_freq=power_bench$frequency-min(power_bench$frequency)
proc_mod=lmer(meanpower ~ shift_freq +(shift_freq | processor),
              data=power_bench)

dp_shift=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

# dotplot comes from the lattice package which uses grid layout
grid.arrange(dp_orig$processor, dp_shift$processor, nrow=2)
```

The upper plot, in Figure 11.56, is the model fitted using the original data; the intercept (upper left) and slope (upper right) appear to be correlated. Looking at the straight line fits for each processor in Figure 11.54, they appear to share an origin starting at the lowest frequency measured; an intercept included as a random effect has a common origin assumed to start at zero (see Figure 11.55). Shifting frequency values down, by the minimum measured value, and refitting a model produces the confidence intervals in the lower plot. The correlation has disappeared; perhaps including the intercept as a random effect is not worthwhile.

Refitting a model without the intercept as a random effect, produces a model that differs from previous models by a small amount (see [regression/hotpower-mix-plot](#)).

There is an upper limit on the number of random effects (i.e., number of unknowns) that can occur in a model. The total number of unknown random effects must be less than the number of observations, otherwise the equations do not have a unique solution. A continuous explanatory variable counts as a single unknown, while a variable holding nominal or ordinal values contributes one unknown for each of the possible discrete values (there is no slope associated with fitting a variable that is not treated as being continuous).

The bootstrap can be used to calculate confidence intervals for a mixed-effects model.

## 11.7 Generalised Additive Models

The regression modeling techniques discussed so far have required the analyst to specify an equation expressing the detailed relationship between explanatory variables and the response variable (these are said to be *parametric models*). If no equation provides a reasonable fit, or accuracy of prediction is important (rather than understanding), then a *Generalised additive model* (GAM) is an alternative approach. A GAM only requires a list of explanatory variables and a response variable to be specified (these are said to be *nonparametric models*).

A GAM is built by finding the best fit for a sequence of polynomial equations (e.g., some form of spline), that smoothly captures the shape of the data. These smooth equations might be used to make predictions, or when the fitted model is plotted may suggest possible parametric

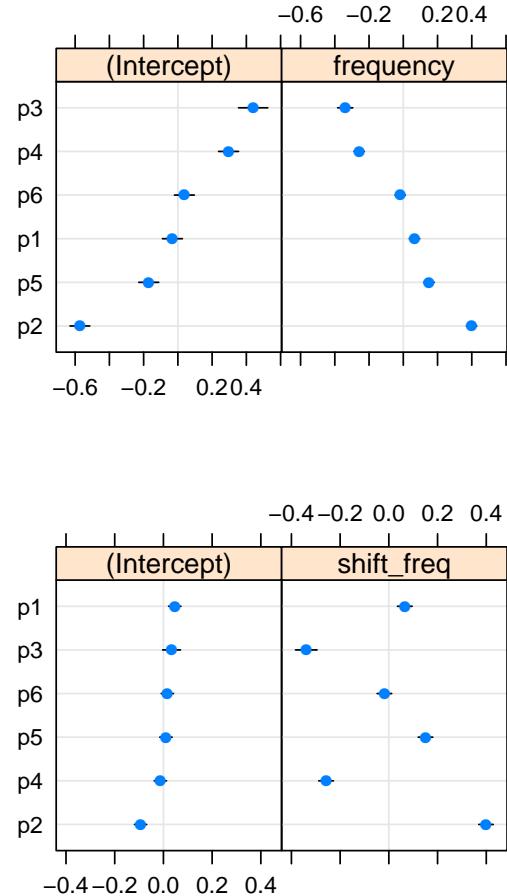


Figure 11.56: Confidence intervals, 95%, for within-subject intercept and slope (right plots) of mixed-effects models in the adjacent code. [code](#)

equations. The details of the fitted equations are not a source of understanding, but they may make good predictions.

The `gam` function, in the `mgcv` package, can be viewed as extending the functionality of `glm` to support a variety of nonparametric smoothing functions (the `gam` package is simpler, but does not offer such a wide range of functionality). The following code shows formulas using a potentially different smoothing polynomial for each explanatory variable (first line below), a different smoothing polynomial for some combinations of explanatory variables (second and third line), a combination of a smoothing polynomial and parameterised form (fourth line) or an interaction between a smoothed and non-smoothed variable (fifth line; the `by` parameter, rather than the `:` operator is used):

```
mod=gam(y ~ s(x_1) + s(x_2) + s(x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3) + s(x_3, x_4) + s(x_4), data=foo_bar)
mod=gam(y ~ x_1 + s(x_2) + x_3, data=foo_bar, family="poisson")
mod=gam(y ~ x_1 + s(x_2, by=x_1) + x_3, data=foo_bar, family="poisson")
```

The smoothing function, `s`, supports a variety of options for controlling the fitting process; two that are likely to be encountered are `k`, which specifies an upper limit on the degrees of freedom that can be used in the fitted polynomial, and `bs` a string identifying the kind of smoother (e.g., "tp", the default, for a thin plate regression spline and "cr" for a cubic regression spline).

The value of `k` needs to be large enough to support the degrees of freedom needed by a polynomial capable of representing the underlying pattern in the data; the `gam.check` function provides information about fitted models that can be used to help select a value for `k`.

The fitting procedure used, by the `mgcv` version of `gam`, tries to avoid overfitting by making every degree of freedom pay its way (using, for instance, *penalized regression splines*). Criteria used for measuring the *cost-effectiveness* of more complicated models include generalised cross-validation (GCV; the default) and AIC. The `select` argument provides support for *null space penalization*, see package documentation for details.

A study by Lee and Brooks<sup>934</sup> built a model to predict the performance and power consumed by applications running on processors having various hardware configurations, e.g., number of registers, size of cache and instruction latency.

The following additive model is based on the one proposed by Lee et al and explains over 95% of the variance in the data (see [regression/lee2006.R](#)). While this model is likely to be useful for prediction, it provides virtually no insight into the impact of various hardware attributes on performance characteristics.

```
l_mod=gam(sqrt(bips) ~ benchmark + fix_lat
           +s(depth, k=4) + s(gpr_phys, k=10)
           +s(br_resv, k=6) + s(dmem_lat, k=10) +
           s(fpu_lat, k=6)
           +s(l2cache_size, k=5) + s(icache_size, k=3) +
           s(dcache_size, k=3)
           +s(depth, gpr_phys, k=10)+s(depth, by=width, k=6)
           +s(gpr_phys, by=width, k=10)
           , data=lee)
```

The analysis around Figure 8.32 used two approaches to modeling the number of accesses to a function's local variables. Without knowing anything about what relationships might exist between explanatory and response variables, and being willing to use very high degree polynomials, it is possible to build and use `gam` to build a prediction model

In the calls to `gam` below, the first assumes there is an interaction between the two explanatory variables (allowing up to 75 degrees of freedom) and the second assumes the variables are independent (allowing up to 50 degrees of freedom for each of them). While the fitted model might make usable predictions (see [sourcecode/local-use/obs-fit.R](#)), the use of such high degree polynomials suggests that the underlying processes have a non-polynomial form.

```
locg_mod=gam(norm_occur ~ s(object.access, total.access, k=75),
              data=common_loc, family=Gamma)
```

```
locp_mod=gam(norm_occur ~ s(object.access, k=50)+s(total.access, k=50),
              data=common_loc, family=Gamma)
```

## 11.8 Miscellaneous

Issues that have no other obvious home...

The p-value, for the coefficients of a fitted model, is a test of the hypothesis that the coefficient is zero, i.e., there is no association. When the actual value of a coefficient is close to zero, the reported p-value may be spurious. One solution is to rotate the axes, which will have the effect of increasing the value of the coefficient and removing this artefact from the p-value calculation (for this data).

### 11.8.1 Advantages of using `lm`

This book promotes `glm` as a one stop solution, however, the `lm` function has some advantages over `glm`, including:

- requiring less cpu time to fit a model. If many models need to be fitted on a regular basis, the performance difference may be worth considering,
- requiring less memory to fit a model. For extremely large datasets, memory requirements may be excessive for `glm`; possible solutions that continue to use `glm` are discussed below,
- the algorithm used by `lm` is always guaranteed to converge to a solution, singularities generated by correlation between explanatory variables excluded. There are edge cases where `glm` does not find a solution without being given some reasonable starting values.

The implementation of `lm` is based on the mathematics of *Ordinary Least Squares* (OLS) and the data has to satisfy additional conditions for OLS to be applicable. Perhaps the most important new condition is that the measurement error variance be constant (in practice close to constant). The `ncvTest` function, in the `car` package, checks that a fitted model meets this requirement; the `spreadLevelPlot` function provides some visualization. Also, see the `lmtest` function.

A user interface issue with models fitted using `glm` is that they do not come with a scale-invariant goodness of fit number, i.e., the R-squared value.

### 11.8.2 Very large datasets

The `biglm` package supports fitting regression models using data that is too large to fit in memory all at once; the models are built using an incremental algorithm, which only requires a subset of the data to be held in memory at any time. A variety of options are available for creating chunks of data to feed into the model building process, including incremental reading from files and databases.

The `biganalytics` package extends the `bigmemory` package by providing interfaces to various analytic packages, such as `biglm` (see [benchmark/bounds\\_chk.R](#))...

The `glm` function stores a lot of information in the object it returns, much of which is often not subsequently used...

### 11.8.3 Alternative residual metrics

The error metric used by many regression techniques is based around squaring the difference between the actual and predicted value. This choice has been driven by the usefulness of the mathematical properties of sum-of-squares. Other error metrics could be used to fit models, e.g., the absolute difference between actual and predicted values.

The `rlm` function, in the `MASS` package, supports analyst specified functions for calculating the residual to be minimised when fitting a model (the `psi`). Supported functions include... Huber...

### 11.8.4 Quantized regression

While removing 1% of outliers might make a significant difference to model accuracy, a quantized regression model might be more informative...

Minimises the residuals of the median (rather than the mean)...?

### 11.8.5 Network data

A regression model can be fitted to network data. But it does require some data to show how...

?

### 11.8.6 Solving systems of equations

systems of unrelated equations, the `systemfit` package...

under-overspecified linear equations or inequality conditions `limSolve` package...

### 11.8.7 Communicating fitted model details

Information about fitted models needs to be communicated to the intended audience. The output produced by `summary` may appear cluttered in some contexts and it is designed for consumption by casual readers.

The `texreg` package contains function for mapping statistical model values to LaTeX and HTML tables...

## 11.9 Item response theory

Item response theory (IRT) deals with the design, analysis and scoring of tests and questionnaires. The `ltm` package supports the analysis of item response data. See Section 13.5 for a discussion of the use of IRT.

There is no guarantee that data can be fitted, using this model. A study by Wohlin, Runeson and Brantestam<sup>1674</sup> investigated the faults found in an 18-page document by student and professional developers. Your author was unable to fit an IRT model to the data; see `regression/stvr95.R`...

### 11.9.1 Agreement between raters

A measurement may be based on human judgement, e.g., assigning a product rating. Different people may make different judgements of the same characteristic/entity, and a way of evaluating the agreement between the different judgements is needed.

Cohen's Kappa is a measure of inter-rater agreement between two raters, it varies from zero (no agreement) to one (perfect agreement). When comparing categorical variables, all differences are usually given equal weight (i.e., considered equally different). When comparing ordinal values, where the degree of difference between raters is easier to quantify, a weighting based on the square of the difference is often recommended...

Fleiss's Kappa is a measure of inter-rater agreement between three or more raters.

The `kappa2` function, in the `irr` package, supports Cohen's Kappa (weighting is supported by passing the argument `weight="squared"`). The `kappam.fleiss` function calculates Fleiss's Kappa.

A study by Schach, Jin, Yu, Heller and Offutt<sup>1379</sup> categorised the kinds of maintenance activity performed on various systems. Table 11.4 lists the categories assigned by two raters to 215 maintenance categories involving the first 20 versions of Linux. The value of Cohen's Kappa for these two raters is 0.805; see `group-compare/agreement.R`.

Rater agreement as a means of measuring ability of rules to be accurately followed<sup>1605</sup>...

	Adaptive	Corrective	Perfective	Other	Total
Adaptive	2	0	0	0	2
Corrective	0	82	16	0	98
Perfective	0	5	99	2	106
Other	0	0	0	9	9
Total	2	87	115	11	215

Table 11.4: Maintenance categories assigned by two raters for the first 20 versions of the Linux kernel at the change-log level. Data from Schach et al.<sup>1379</sup>

## 11.10 Time series

Time series analysis deals with measurements that are sequentially correlated. An example of correlated measurements is current room temperature, which is likely to be similar to the temperature 10 minutes ago and the temperature 10 minutes from now. Techniques developed to analyse time-series can be used to analyse measurements of any quantity, where a correlation exists between successive measurements.

The base system provides basic functions for analyzing time series of continuous values.

A time series contains one or more of the following three components:

- underlying trend: which changes slowly,
- regular recurring pattern of changes (known as *seasonality*): for instance, expected daytime temperature throughout the year,
- random, irregular or fluctuating component.

The `stl` function (Seasonal Trend using Lowess) provides a way of splitting a time series into these three components (the argument must be an object of type `ts`, with a user specified frequency; the `stl` function does not automatically detect the recurrence period), and there is a corresponding plot function.<sup>xxvi</sup>

Figure 11.57, from a study by Eyolfson, Tan and Lam,<sup>462</sup> shows the three time-series components of the hourly rate of commits to the Linux kernel source tree, over the days of a week (the commits during the same hour of the same day were summed). The `stl` function assumes a fixed, recurring, pattern of seasonal behavior, a slowly changing trend, with everything else classified as random noise.

```
# A seasonal frequency has to be specified
hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)
plot(stl(hr_ts, s.window="periodic"))
```

Possible outputs from time-series analysis include:

- a fitted model specifying how the value of a quantity at time  $t$  depends on its values at earlier measurement times (often at  $t - 1$ ),
- a regression model, adjusted for correlation between sequential measurements,
- a power spectrum showing the dominant frequencies present in the data,
- a hierarchical clustering of multiple time series.

Structure is often added to the continuous, linear, nature of time by imposing repeating fixed length intervals, such as hours of the day and days of the week. Many time series analysis techniques require measurements to be made at fixed length intervals; analysis of measurements at irregular intervals is not discussed here (but if we had some data...).

Some library functions use a time series datatype for representing time related measurements. The `ts` function, part of the base system, converts a vector to class `ts` (many time series functions will automatically convert vectors to this class).

The `xplore` function, in the `lattice` package, can be used to create a time series strip chart, see Figure 8.19.

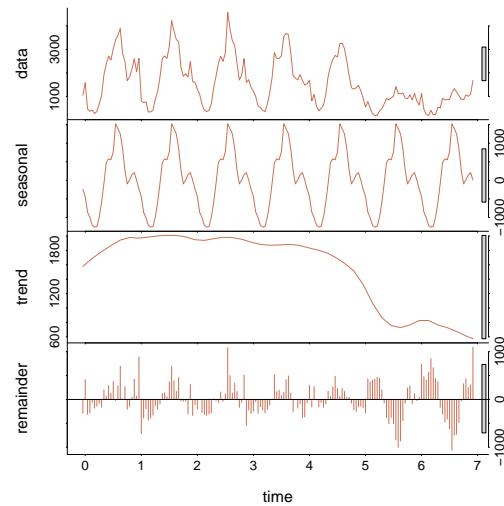


Figure 11.57: The three components of the hourly rate of commits, during a week, to the Linux kernel source tree; components extracted from the time series by `stl`. Data from Eyolfson et al.<sup>462</sup> code

<sup>xxvi</sup> The `decompose` function, part of the base system, implements the same functionality in a less sophisticated way.

### 11.10.1 Cleaning time series data

Many time series techniques implicitly assume that measurement data occurs at regular intervals. A measurement process may only record events when they occur and if no event occurred in within an interval there may be no data-point for that interval. Part of the cleaning process involves ensuring that every interval contains a value (which may be zero or inferred from surrounding values).

A study by Buettner<sup>227</sup> gathered project staffing information for several commercial development software projects. On large commercial projects the amount of work done at weekends is likely to be zero (except for the weeks prior to major deliveries) and the autocorrelation of project activity is likely to show a recurring pattern involving two consecutive days separated by seven days, i.e., weekends and weekdays.

Figure 11.58 shows the autocorrelation of the number of defects found on a given day, for one development project. The seven day recurring pattern contains a three consecutive day pattern, are the developers only working a four-day week? It turns out <sup>xvii</sup> that contractors on some projects work a two-week cycle, with extra hours worked one week and then not working the Friday of the following week. The extent to which regular staffing level differences, between Friday and other weekdays, has to be taken into account, will depend on the kind of analysis performed (weekends can be handled by excluding them from the analysis, focusing on where most effort occurs, i.e., week days).

Measurements made on public holidays, such as the New Year, are very likely to differ from normal work days. Removing public holidays from the data will scramble the association with day of the week. The extent to which day of the week is a more important factor in the analysis, than public holidays, has to be considered.

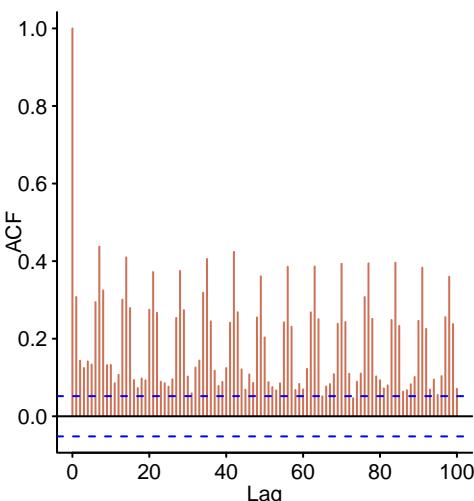


Figure 11.58: Autocorrelation of number of defects found on a given day, for development project C. Data kindly provided by Buettner.<sup>227</sup> [code](#)

### 11.10.2 Modeling time series

The expected mean of a time series can be modeled using one or both of the following two approaches (series whose variance is serially correlated are discussed later):

- the *Autoregressive model* (AR), models the value at time  $t$  as a weighted combination of values from earlier time steps, plus some amount of added noise ( $w_t$ ), for instance:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t$$

is an autoregressive model of order 2, abbreviated AR(2); it is based on values going back two time steps (with weights  $\phi_1$  and  $\phi_2$ ).

The `ar` function fits data to an autoregressive model.

- the *Moving Average model* (MA), models the value at time  $t$  as the sum of noise ( $w_t$ ) and a weighted combination of the noise from earlier time steps, for instance:

$$x_t = w_t + \theta_1 w_{t-1}$$

is a moving average model of order 1, abbreviated MA(1); it uses a value from one time step back (with weight  $\theta_1$ ).

The `arima` function, with the first two values of the order argument set to zero, fits data to a moving average model.

The autocorrelation function, `acf`, returns and plots the correlation of a time series with itself at successive lag intervals (i.e., the correlation of the measurement at time  $t$  with the measurement at time  $t+n$ ; the default sequence of lags is  $n=1:25$ ); see Figure 11.59. For the AR(1) model,  $x_t = \phi x_{t-1}$ , the impact of serial correlation on values separated by  $k$  lags (time intervals) decreases by  $\phi^k$ .

The lag 0 autocorrelation is always one and the two dotted blue lines are 0.05 p-value bounds. Each lag is a hypothesis test, and with 25 hypothesis tests (the default) at least one calculated value is expected to exceed a 0.05 p-value with probability  $1 - 0.95^{25} \rightarrow 0.72$ ; also, successive measurements are correlated, so neighbouring lag points are likely to show similar significance levels.

The partial autocorrelation function (the `pacf` function) calculates and plots the correlation at lag  $k$ , after removing the effect of any correlation generated by terms at shorter lags; see Figure 11.60. The partial autocorrelation at lag  $k$  is the  $k^{\text{th}}$  coefficient of an AR( $k$ ) model.

<sup>xvii</sup> Email discussion with Buettner.

The previous two plots illustrate how short range correlations in an AR model have a long range impact on the values returned by acf, but an MA model does not have a long range impact, while the opposite behavior is seen in the values returned by pacf. An ARMA model always behaves in the most unhelpful way.

An ARMA model (*Autoregressive Moving Average*) is a combination of an AR and MA model, e.g., ARMA(2, 1) is the sum of an AR(2) and MA(1) model, such as the following:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t + \theta_1 w_{t-1}$$

The ARMAacf function takes a specification of an ARMA model, and returns what acf would return when passed a time series following this model (the pacf=TRUE option switches the behavior to that of pacf).

A time series is said to be *stationary* if the expected mean value does not change over successive measurements, i.e.,  $E[t_i] = E[t_{i+k}]$ , or in conceptual terms, the probability of events driving a stationary process do not change over time. The mathematics behind both the AR and MA models assume a stationary time series.

Many software engineering processes are driven by non-stationary components, e.g., varying number of developers working on a project, more customers, system updates, etc.; significant changes can have an impact large enough for measured values to have a non-stationary time series. ARIMA (*Autoregressive Integrated Moving Average*) handles non-stationary time series (supported by the arima function).

Time series analysis is not limited to data involving time, it can be applied to any data that contains serial correlation between measurements.

A study by Hindle, Godfrey and Holt<sup>703</sup> measured the indentation of the first non-whitespace character on a line, for code written in a variety of languages. Figure 11.61 shows the autocorrelation of a list, ordered by indentation, of the total number of lines having a given indentation.

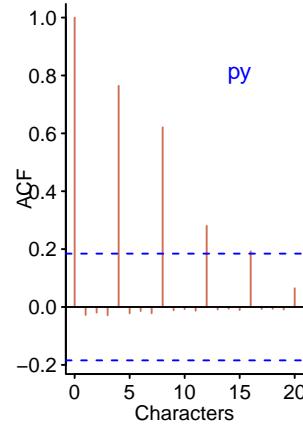
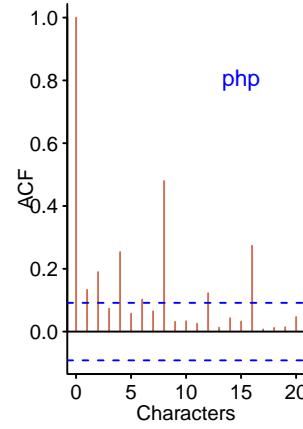
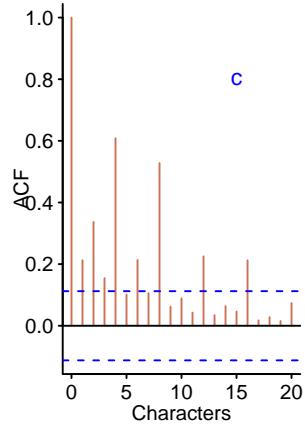
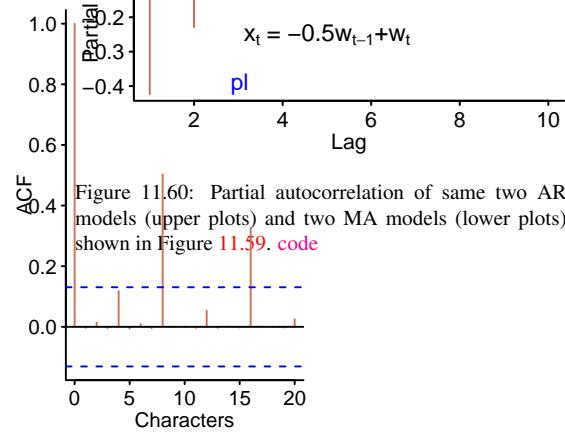
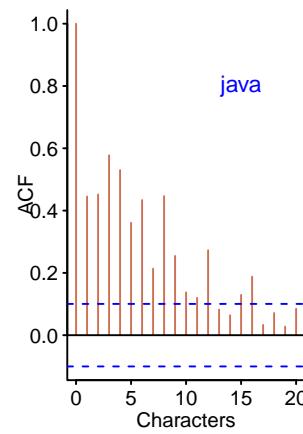
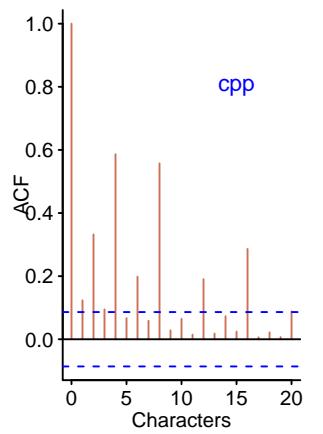


Figure 11.60: Partial autocorrelation of same two AR models (upper plots) and two MA models (lower plots) shown in Figure 11.59. [code](#)

### 11.10.2.1 Building an ARMA model

ARMA modeling takes as input a time series; if this time series is non-stationary, it has to be converted to a stationary form before model building can begin. Common reasons for a time series not being stationary and possible transforms to a stationary series include:

Figure 11.61: Autocorrelation of indentation of source code written in various languages. Data from Hindle et al.<sup>703</sup> [code](#)

- a non-zero trend: for instance, the following equation contains an increasing time dependent trend:

$$x_t = \alpha + \beta t + w_t$$

Differencing can be used to remove trends, but care needs to be taken because this can introduce signals that are not in the original data. For instance, differencing the above equation gives:

$$\Delta x_t = x_t - x_{t-1} = b + w_t - w_{t-1}$$

an MA(1) process which the original series does not contain.

Subtracting the trend  $\alpha + \beta t$  leaves just  $w_t$ ; see Figure 11.62,

- seasonality: this is a cyclic trend, e.g., changes recurring every year. Implementations of ARMA often include support for including a seasonal component in the model, e.g., the seasonal to the arima function,
- non-constant variance (known as *volatility* in the analysis of financial time series).

If the growth in variance, over time, approximately follows the growth of the mean (such as when there is a relatively consistent percentage change at each time step, e.g.,  $y_t = (1 + x_t)y_{t-1}$ ), then a log transform produces a time series with approximately constant variance (e.g.,  $\Delta(\log y_t) \approx x_t$ , assuming  $\log(1 + x_t) \approx x_t$ ).

A log transform can only be applied when values are greater than zero; in the case of the 7digital data (which has an increasing variance, as more developers are employed and time to implement features decreases) there are many zeroes and adding a tiny amount to every value allows a log transform to be made.

The plots produced by acf and pacf provide useful information about the likely structure and order of an ARMA model.

- if the plot produced by acf shows a decreasing trend, while the pacf shows a sharp cut-off (see Figure 11.59), an AR model is a good place to start,
- if the plot produced by acf shows a sharp cut-off, while the pacf shows a decreasing trend (see Figure 11.60), an MA model is a good place to start,
- if both plots show a decreasing trend, then some combination of AR and MA model is likely to be needed.

```
# Add 1e-5 to handle zero values; selection by wet finger process
acf(diff(log(weekdays+1e-5)), xlab="Lag (working days)")
pacf(diff(log(weekdays+1e-5)), xlab="Lag (working days)")
```

Models fitted, by calling arima with various values of its order argument, can be compared using AIC (arima only returns the series mean, when a difference value of zero is passed to order; the Arima function, in the forecast package, is not limited in this way). The auto.arima function, in the forecast package, can be used to automatically find ARIMA model values that minimise AIC.

```
library("forecast")
```

```
arima(diff(log(weekdays+1e-5)), order=c(5, 0, 1))
auto.arima(weekdays, max.order=7)
arima(diff(log(weekdays+1e-5)), order=c(1, 0, 2))
```

The two best fitting models, for the feature start data, are ARMA(5, 1) and ARMA(1, 2). The output from the last call to arima above is: [code](#)

Call:

```
arima(x = diff(log(weekdays + 1e-05)), order = c(1, 0, 2))
```

Coefficients:

	ar1	ma1	ma2	intercept
s.e.	0.8599	-1.6708	0.6708	0.0070
	0.0399	0.0576	0.0574	0.0014

```
sigma^2 estimated as 19.66: log likelihood = -2492.11, aic = 4994.23
```

**The Coefficients:** table lists the model coefficients and their standard error. The intercept column is the mean value of the time series (which for a stationary series is zero). The equation for one of the models is:

$$x_t - 0.0070 = 0.8599(x_{t-1} - 0.0070) + w_t - 1.6708w_{t-1} + 0.6708w_{t-2}$$

which simplifies to:

$$x_t = 0.0070 \times (1 - 0.8599) + 0.8599x_{t-1} + w_t - 1.6708w_{t-1} + 0.6708w_{t-2}$$

with the constant increment per time step calculated as 0.0009807.

Which of these two models provides the better explanation of the data? Features take different amounts of time to implement, and work can only start on a new feature when enough people have been freed up, through completion of work on other features. The coefficients of the AR component, of the ARMA(5, 1) model, can be interpreted as a probability that people working on a feature started a given number of days earlier will become available to start work on a new feature (see Table 11.5).

	AR	Duration
ar1	0.19	0.32
ar2	0.11	0.16
ar3	0.09	0.11
ar4	0.07	0.07
ar5	0.10	0.05

Table 11.5: AR coefficients of ARMA(5, 1) model and percentage of features taking a given number of days to implement. Data kindly supplied by 7Digital.<sup>1</sup> [code](#)

This may be a just-so story, but stories are useful tools and your author cannot think of one for the ARMA(1, 2) model.

**Handling seasonal trends** A seasonal ARIMA model can include AR, difference and MA components at an offset equal to the number of measurement intervals in the season. By default, the `auto.arima` function, in the `forecast` package, will return seasonal components (if any are found). The `seasonal` option can be used to specify seasonal components to the `arima` function.

The following code estimates a seasonal ARIMA model, for hourly commits to the Linux kernel source tree (see Figure 11.57):

```
library("forecast")

hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)

auto.arima(hr_ts)
arima(linux_hr, order = c(2,1,1), seas = list(order = c(1,0,1), period=24))
```

The coefficients of the first and second fitted models, below, differ because of differences in the algorithms used by the functions that fitted them, but are within each other's standard error (the third set of coefficients is for a slightly simpler model): [code](#)

```
Series: hr_ts
ARIMA(2,1,1)(1,0,0)[24]
```

Coefficients:

ar1	ar2	ma1	sar1
-0.9190	-0.3920	0.5022	0.6991
s.e.	0.2059	0.0983	0.2186
			0.0639

```
sigma^2 estimated as 141673: log likelihood=-1233.55
AIC=2477.1   AICc=2477.48   BIC=2492.69
```

Call:

```
arima(x = linux_hr, order = c(2, 1, 1), seasonal = list(order = c(1, 0, 1),
    period = 24))
```

Coefficients:

ar1	ar2	ma1	sar1	sma1
-0.8124	-0.2862	0.4483	0.8909	-0.4516

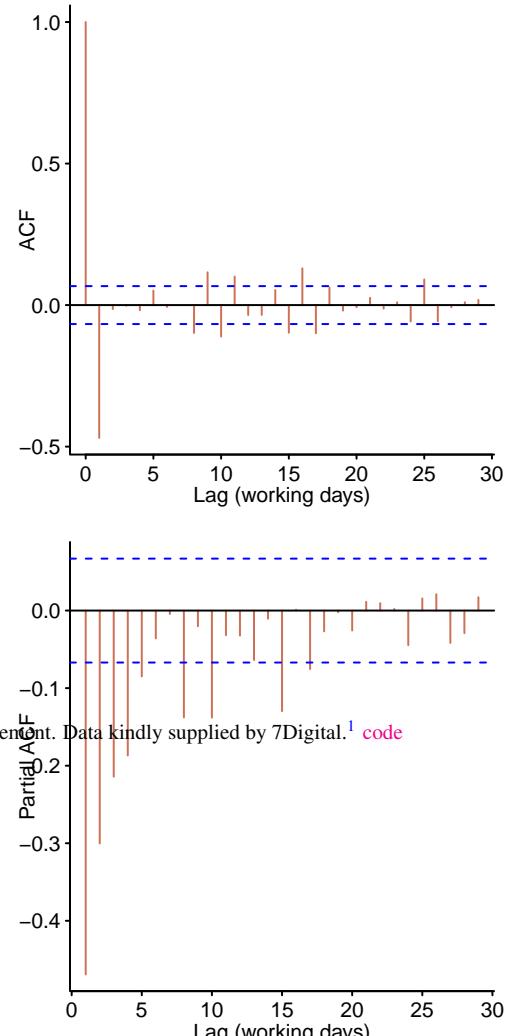


Figure 11.63: Autocorrelation (upper) and partial autocorrelation (lower) of the number of features started on a given day (after differencing the log transformed data), over the entire period of the 7digital data. Data kindly supplied by 7Digital.<sup>1</sup> [code](#)

```
s.e.    0.2995   0.1177   0.3058   0.0546   0.1404
sigma^2 estimated as 129070:  log likelihood = -1229.02,  aic = 2470.03
Call:
arima(x = linux_hr, order = c(2, 1, 0), seasonal = list(order = c(1, 0, 1),
period = 24))

Coefficients:
            ar1      ar2      sar1     sma1
-0.3632  -0.1174   0.8980  -0.4727
s.e.    0.1007   0.0964   0.0519   0.1391
sigma^2 estimated as 129942:  log likelihood = -1229.71,  aic = 2469.42
```

The `sar1` is the seasonal AR coefficient and `sma1` the seasonal MA coefficient.

The output from `auto.arima` is a suggested model. In this case the `ar` and `sar` coefficients are pulling in opposite directions and the standard error for the `ma1` coefficient is very high. Removing the MA component produces a model (second call to `arima` above), where the coefficients are not almost cancelling each other out; the model is (24 is the seasonal period):

$$x_t = -0.4x_{t-1} - 0.1x_{t-2} + 0.9x_{t-24 \times 1} - 0.5w_{t-24 \times 1}$$

What happened 24 hours ago is a contributes more to the predictor than what happened in the previous hour or two.

**Predictions made using a fitted ARMA model** A fitted ARIMA model can be used to predict what may occur after a measurement at time  $t$ ; the relatively large noise component present in some ARMA models means that the confidence bounds of the predicted values may quickly become very wide. The base system supports a `predict` function that accepts models fitted by the `arima` function; the `Arima` and `forecast` functions, from the `forecast` package, support more options for fitting and forecasting of time-series data. In the following code, the `include.drift=TRUE` option specifies that trend information be included in the model; for this data, the increasing volume of sales generates an increasing trend:

```
library("forecast")
Ar_mod=Arima(data$Spreadsheets, order=c(1, 0, 1), include.drift=TRUE)
f_pred=forecast(Ar_mod, h=10)
plot(f_pred, col=point_col, main="", xaxs="i",
      xlab="Month", ylab="Monthly sales\n")
```

A study by Givon, Mahajan and Muller<sup>574</sup> investigated UK sales of PC's, wordprocessors and spreadsheets. Figure 11.64 shows monthly sales of spreadsheets and, based in an `arima` model, 12-months of predicted sales; shaded areas are 80% and 95% confidence intervals.

### 11.10.3 Non-constant variance

Variance in time-series data is primarily of interest when generating data for simulation. Time-series data that experiences rapid changes in variance is said to be *volatile*; correlated variance is common during periods of volatility (a time series is *heteroskedastic* if the change in variance is regular and *conditionally heteroskedastic* if the change is irregular). Techniques for building an autoregressive model, for the variance, include *autoregressive conditional heteroskedastic* (ARCH) and *generalised ARCH* (GARCH) models.

A time-series experiencing changes in variance is not stationary.

An increase in frequency of commits leading up to a major new release is an example of behavior that can cause a change of variance in a time series.

The autocorrelation of a time-series may not show any correlation, but if its variance changes the square of the zero adjusted values will have a pattern of decreasing correlation in its acf; see Figure 11.65.

```
acf(t_series)
acf((t_series-mean(t_series))^2) # Check for changing variance
```

The `garch` function, in the `tseries` package, fits a GARCH model to data.

### 11.10.4 Long-memory processes

Correlation at high lags... just need some data... Fractal nature of LAN traffic... (is not SE)

Fractionally differences ARIMA processes (FARIMA)...

The `fracdiff` package...

### 11.10.5 Smoothing and filtering

Smoothing a time-series can make it easier to visually identify larger scale patterns and also provides a simple approximation to predicting immediate future values. Even when data does not contain a systematic trend, or seasonal effects (perhaps because they have been removed), it may still be possible to make a useful estimate of immediate future values based on immediate past values.

Smoothing using the *exponentially weighted moving average* (EWMA; also known as *exponential moving average*, EMA) uses the following formula:

$$EMA_t = \phi x_t + (1 - \phi)EMA_{t-1}$$

where:  $x_t$  is the measured value of  $x$  at time  $t$ , and  $\phi$  determines the amount of smoothing. The *exponential moving standard deviation* (EMS) is given by:

$$EMS_t = \sqrt{\phi EMS_{t-1}^2 + (1 - \phi)(x_t - EMA_t)^2}$$

EMA and EMS can be used to detect when a real-time data stream trends outside of pre-specified bounds.

Holt-Winters smoothing is a generalization of exponential smoothing, that uses three parameters: estimated level, slope and seasonality; the `HoltWinters` function can be used to both estimate and apply these parameters.

A call to `plot` will display the three components of a time series based on the value returned by the `HoltWinters` function... Seasonal component can vary...

The `filter` function can be used to apply a linear filter to a time series.

TODO some data...

### 11.10.6 Missing data

Handling missing data in time-series can be even more complicated and difficult than in regression modeling.

A study by Buettner<sup>227</sup> gathered data on project staffing, but was not always able to obtain complete staffing information. Figure 11.66 shows a loess fit to the available data along with the standard error...

Some R functions support the use of splines for interpolation. Splines originated as a method for connecting a sequence of points by a smooth curve, not as a method for fitting a curve that minimises some error metric. Apart from their familiarity, there is no reason to prefer the use of splines over other techniques (implementation issues also exist with the `bs` and `ns` functions, in the `splines` package, when fitting a model with the `predict.glm` function<sup>1600</sup> and then making predictions using new data points)...

### 11.10.7 Spectral analysis

A series of measurements in the time domain can be transformed into the frequency domain; see Figure 1.12.

The `spectrum` function estimates the spectral density of a vector, which is assumed to be a time-series (the default behavior is to call the `spec.pgram` function). The `spec.arma` function takes a specification of an ARMA model and returns its power spectrum, i.e., behaves like a call to `spectrum` when passed a time series that follows this model.

A stationary time-series does not contain components at specific frequencies, but it can be described in terms of an average frequency composition.

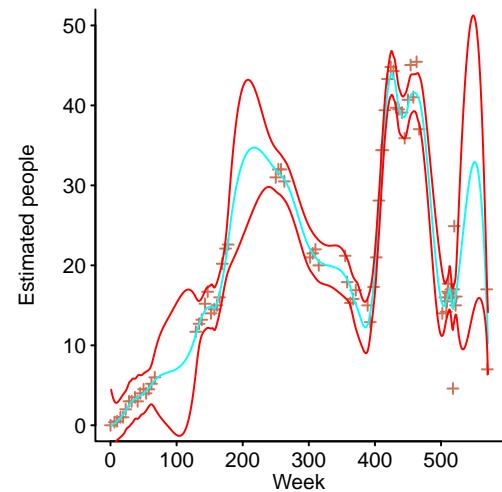


Figure 11.66: Estimated staff working on a project during every week. Data from Buettner.<sup>227</sup> [code](#)

### 11.10.8 Relationships between time series

Some of the relationships that can exist between two or more time-series include:

**Cross-correlation:** The correlation, at various lags, between two stationary time-series. Figure 11.67 shows the cross-correlation between the number of source lines added/deleted, per week, to the glibc library. In calls to the `ccf` function, the first argument is the one which is shifted, while the second is fixed. In the following call:

```
ccf(lines_added, lines_deleted, col=point_col, xlab="Weeks")
```

the plot produced shows correlation spikes, above the confidence bounds, occurring between the sequence pairs  $\text{lines\_added}_{t+2}/\text{lines\_deleted}_t$  and  $\text{lines\_added}_{t+8}/\text{lines\_deleted}_t$  (i.e., changes involving  $\text{lines\_deleted}$  is correlating with changes to  $\text{lines\_added}$  two and 10 weeks later; a positive lag means the first argument follows the second, a negative lag that it leads the second); there are small spikes at:  $\text{lines\_added}_{t+8}/\text{lines\_deleted}_t$  and  $\text{lines\_added}_{t-13}/\text{lines\_deleted}_t$ . Your author has no explanation for this correlation.

**Alignment:** A time series is a sequence of values, with each value being larger, smaller or equal to the value immediately before it. If two time series are generated by the same, or similar, process they may contain subsequences of values that share the same pattern of up, down and no-change. A non-time series application of this kind of subsequence matching, is locating word sequences common that appear in two documents.

Dynamic time warping (DTW) is a class of algorithms that compares two series of values by stretching or compressing one of them (treated as the reference series), so it resembles the other (treated as the query series). The `dtw` package contains functions to perform and support DTW alignment of two series.

A study by Herraiz<sup>1529</sup> investigated the evolution of various long-lived software systems, and measured the growth of NetBSD and FreeBSD (in lines of code). These two operating systems started from the same base, continue to share developers (see Figure 9.21) and code continues to be ported between them. Figure 11.68 shows the alignment, found by a call to `dtw`, between the weekly measurements of the lines of code in each OS (for the first 100 weeks of their development).

```
library("dtw")

bsd_align=dtw(freebsd_weeks, netbsd_weeks, keep=TRUE,
              step=asymmetric, open.end=TRUE, open.begin=TRUE)
plot(bsd_align, type="twoway", offset=1, col=pal_col, xlab="Weeks")
```

**Clustering:** Many techniques for measuring the distance between two time-series have been invented (at the time of writing, the `diss` function, in the `TSclust` package, supports 22 distance metrics). The pair-wise similarity of multiple time-series can be used as a clustering metric. The `TSclust` package contains functionality for the clustering of time-series.

A study by Powell<sup>1274</sup> investigated task effort allocation in a development project at Rolls-Royce. Figure 11.69 shows effort (in person hours) spent on eight major tasks (lower plot, from the bottom up: s/w requirements, top-level design, coding, low level test, requirement test, system acceptance test, management and holiday/non-project) and a hierarchical clustering of each task by its effort time series, with pair-wise distance between time series calculated using correlation (upper) and Euclidean (middle) metrics.

```
library("TSclust")

eff_dist=diss(t(all_effort), METHOD="COR")
plot(hclust(eff_dist), main="", sub="", xlab="", ylab="Correlation distance")
```

### 11.10.9 Regression models

The mathematics underpinning many regression modeling techniques requires each measurement in a sample to be independent of the other measurements in that sample. A sample of time-series data is often serially correlated.

Some regression modeling functions can adjust for the presence of serial correlation (information about the correlation is passed in an optional argument). The `gls` function, in the

`nlme` package, supports a correlation option; the `dynlm` package supports the use of time series operators (e.g., `diff` and `lag`) in the specification of model formula; the `tscount` package supports the fitting of generalized linear models to time series of count data.

[time-series/agile-week-acf.R...](#)

The number of source lines in FreeBSD is growing over time, see Figure 11.2...

### 11.10.10 Stochastic processes

The `Sim.DiffProc` package can be used to numerically solve stochastic differential equations of the Itô type...

The *Ornstein-Uhlenbeck process* is the continuous time version of an AR(1) model.<sup>408</sup> The AR(1) process corresponding to the equation:

$$dx = \eta(\hat{x} - x)dt + \sigma dz$$

where:  $\eta$  is the speed of reversion, and  $\hat{x}$  is the mean value, is:

$$x_t - x_{t-1} = \hat{x}(1 - e^{-\eta}) + (e^{-\eta} - 1)x_{t-1} + \varepsilon_t$$

The parameters of an AR(1) model fitted to the available data can be used to calculate  $\sigma$  and  $\eta$ .<sup>408</sup>

??

### 11.10.11 Misc

Outlier detection...

The `cpm` package supports change-point analysis of time series having a variety of distributions, e.g., exponential, poisson and the `ecp` package supports change-point analysis of multivariate time series...

Granger causality...

## 11.11 Survival analysis

Survival analysis is the analysis of data where the response variable has the form of *time-to-event*. Historically this kind of model building has been used to compare the impact of different medical procedures, or drugs, on subject survival rate.

Often, survival analysis deals with one kind of event, which causes a transition to a terminal state, e.g., there is returning from the dead. Competing risk models deal with the situation where one of several risk events can cause the transition to the final state. Multistate models handle the situation where some transitions are to states that are not final, i.e., an appropriate event can cause a transition to another state.

In some cases, the event of interest may not occur during the measurement period, in this case the measurement is said to be *censored*; for instance, when measuring the time interval between a function definition being written and the first time it is modified, the measurement data is said to be *right censored*, when one or more functions have not been modified over the time interval for which data is available.

Survival analysis makes greater use of the available information, to produce estimates containing less error, than other forms of regression modeling; a linear regression model comparing mean time-to-event between groups would have to ignore censored data, while a logistic regression model, using 0/1 to indicate whether a subject survived or not, would again have to ignore censored data.

Possible outputs from survival analysis include:

- a survival function,  $S(t)$ , the probability of surviving a given amount of time. This can be used to estimate time-to-event for a group of subjects or compare time-to-event between subjects in two or more groups,

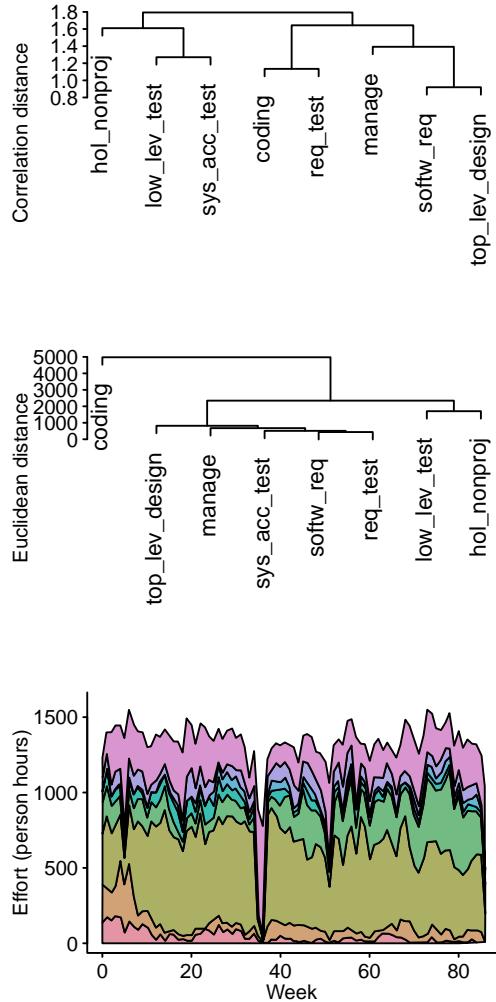


Figure 11.69: Effort distribution (person hours) over the eight main tasks of a development project at Rolls-Royce and a hierarchical clustering of each task effort time series based on pair-wise correlation and Euclidean distance metrics. Data extracted from Powell.<sup>1274</sup> [code](#)

- a hazard function,  $h(t)$ , the hazard rate, that is, the probability of an entity surviving to time  $t$  experiencing an event in the next time interval, e.g., having survived 69 years 11 months before reading this sentence the probability that you die in the next month (the interval used to denote an instant is small compared to the time spans involved). The survival and hazard functions can be derived from each other:

$$h(t) = \frac{f(t)}{S(t)}$$

where:  $f(t)$  is a probability density function, the probability of the event occurring at exactly  $t$  time units in the future, e.g., the probability of a baby born 70 years ago living long enough to read this sentence, but not before,

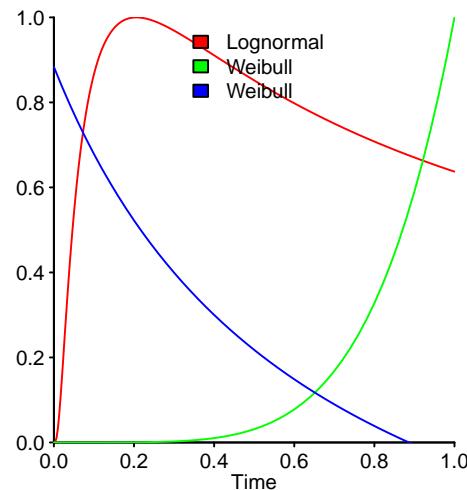


Figure 11.70: Two commonly used hazard functions; Weibull is monotonic (always increases, decreases or remains the same, depending on the equation coefficients) and Lognormal which can increase and then decrease.

[code](#)

- a regression model specifying the impact of explanatory variables on time-to-event. This may be a non-parametric model, such as the Cox proportional hazard model, because parametric models can be very difficult to build.

Time-to-event is always positive and so has a skewed distribution (which means it cannot have a Normal distribution).

The `survival` package contains functions implementing the functionality needed to perform survival analysis.

### 11.11.1 Kinds of censoring

Ideally censoring is uninformative, i.e., the distribution of censoring times provides no information about the distribution of survival times. When a period of study is decided in advance, the censoring information is uninformative.

When censoring is not under the control of the experimenter, it is said to occur at random. For instance, a subject may decide to stop taking part in a study because they are not happy with their performance.

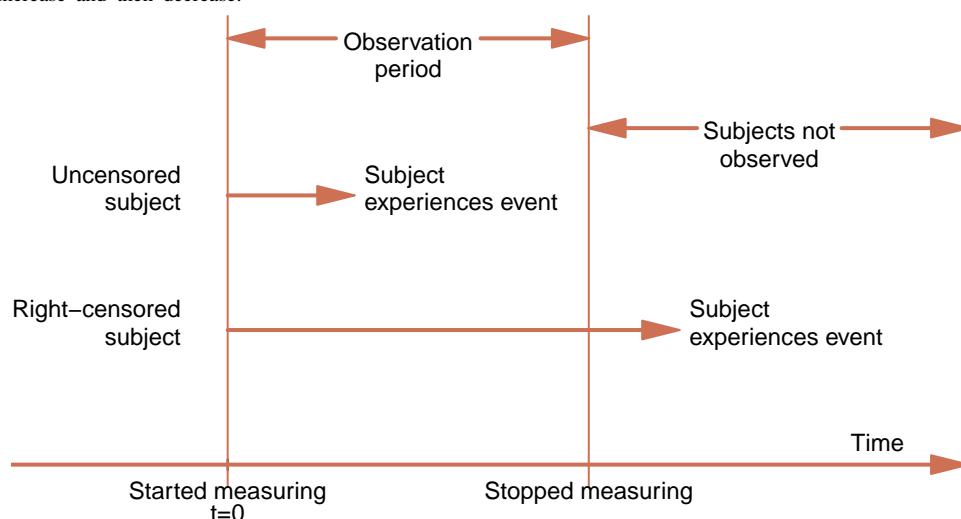


Figure 11.71: Observation period, with events inside and outside the study period. [code](#)

Kinds of censoring that can occur include:

- *left truncation*: subject not observed before  $t_0$ , experienced an event before that time and is not included in the study (the event may have been such that it rendered the subject unable to join the study, e.g., developer left the company),
- *left censored* (also *left truncated*): a subject included in the study is known to have had the event prior to time  $t$ , but with the exact time not being known,
- *right censored*: described at the start of the subsection,
- *interval censored*: when measurements are made at regular intervals, the exact time of an event is not known, only that it occurred between two measurement points,
- *non-detect*: the measurement process may fail to detect an event because the strength of the event is below the detection threshold. This kind of censoring is not covered here, see Helsel.<sup>677</sup>

### 11.11.1.1 Input data format

The `Surv` function creates a survival object from data, and the object it returns plays the role of the response variable in formula passed to model building functions. The required data format depends on the kind of censoring and presence of time dependencies. The following is an example of the basic information required:

```
id,start_time,end_time,failure_status,explanatory_v1,explanatory_v2
```

where: `id` is a unique identifier denoting each subject (only needed when information on the same subject occurs on multiple lines), `start_time/end_time` the starting time (or date) of measurement and the end time (either when the event occurred, the end of the study or the last recorded time of a subject who was not seen again) and `failure_status` one of two values specifying whether an event occurred or not; followed by an optional list of explanatory variables.

The time of interest is the difference between the start/end time and the data may contain just this value.

## 11.11.2 Survival curve

The *Kaplan-Meier* curve is a descriptive statistic of time-to-event measurements; it shows the percentage of subjects who have not experienced an event up to a point in time, along with an optional confidence interval (see Figure ??)

The median is preferred over mean as the measure of central tendency for survival data, because the mean underestimates the true value when samples contain censored data. The median is measured as the point where the Kaplan-Meier curve falls before 0.5 and printing the model returned by `survfit` gives this value along with its 95% confidence intervals.

A study by Businge, Serebrenik and van den Brand<sup>236</sup> investigated the number of releases of Eclipse third-party plug-ins (ETP) between 2003 and 2010; the history of each ETP was traced from the year of its first release and any releases in subsequent years were noted.

The Eclipse framework includes a published list of officially recognised APIs, but each release of the Eclipse SDK also includes support for APIs considered to be for internal use, i.e., not to be used by applications. The status difference between official/internal APIs is that internal APIs can be changed without notice, while the official APIs are intended to have some degree of permanence (they may change on major releases but are not intended to change on minor releases; starting in 2004 all yearly releases were minor).

At some point there are no new releases of an ETP in a year and this cessation of new releases could be regarded as the *death* of development of the ETP (some ETP development died for one year only to be resurrected the following year; for simplicity the small number of such recurring events are ignored).

For this analysis ETP yearly release counts are divided into two groups, those that only made use of official APIs and those that made use of one or more internal APIs; Table 11.6 shows the number of ETPs using only the official API.

	2003	2004	2005	2006	2007	2008	2009	2010
<b>2003</b>	35	10	3	1	1	2	0	0
<b>2004</b>		33	4	4	2	2	0	0
<b>2005</b>			41	10	4	3	1	1
<b>2006</b>				61	7	1	0	2
<b>2007</b>					37	12	4	6
<b>2008</b>						38	7	2
<b>2009</b>							25	3
<b>2010</b>								16

Table 11.6: Total number of distinct ETPs released in a year; left column lists year of first release and releases in subsequent years. Data from Businge et al.<sup>236</sup>

Figure ?? shows the Kaplan-Meier curve for ETPs using only official APIs (blue) and ETPs that use internal APIs (red); the dotted lines are 95% confidence intervals. The following is the essential code (calling the `Surv` function to create a survival object, containing time and censored information on each subject, is the first step in most survival analysis using R):

```
library("survival")

api_surv=Surv(all_API$year_end-all_API$year_start,
              event=(all_API$survived == 0), type="right")
api_mod=survfit(api_surv ~ all_API)
plot(api_mod, col=pal_col, conf.int=TRUE, xlim=c(0,7), xlab="Years")
```

The summary function can be used to obtain the values of the survival curve at each time measurement point.

**Comparing two survival curves** Are the two survival curves statistically different? The survdiff function can be used to answer this question. The p-value returned by the call (bottom right) shows that the two survival curves are very unlikely to be the same: [code](#)

```
Call:
survdiff(formula = Surv(year_end - year_start, event = (survived ==
0), type = "right") ~ API, data = all_API)

      N Observed Expected (O-E)^2/E (O-E)^2/V
API=0 381      334     372      3.83      29
API=1 289      260     222      6.41      29

Chisq= 29  on 1 degrees of freedom, p= 7e-08
```

By default, survdiff performs a *log-rank test*, which gives equal weight to all events. Passing the argument rho=1 causes greater weight to be given to earlier events, while the argument rho=-11 gives greater weight to later events. The hazard function is returned by survfit functions when it is passed the argument type="fh".

Why, on average, do new releases of an ETP using internal APIs occur over a greater number of years? Is it because there are changes to the internal APIs that break the ETP, requiring the ETP to be updated to handle the change and a new version released, or is it because authors who use internal APIs are more committed to creating the best possible product and so continue to refine their ETP over more years?

Perhaps, suspecting that changes to the SDK were a significant factor, Businge<sup>235</sup> investigated the source compatibility of ETPs with the Eclipse SDK across releases 1.0 to 3.7 (i.e., releases in every year from 2001 to 2011). Every ETP was built using each of these 11 SDK releases (yes, even SDKs created before an ETP was first released). To allow easy comparison with the ETP analysis above, the following analysis only considers SDK builds released after an ETP was first made available.

Figure 11.73 shows the survival of ETPs' ability to build under Eclipse SDKs released in each successive year. ETPs using internal APIs (red) are much more likely to fail to build (precompiled plug-ins may still function, if they don't call any changed internal API) when a new Eclipse SDK is released, compared to ETPs using only the official APIs (blue).

This analysis suggests that developers using internal APIs in their ETP, are more likely to be forced to release an update, if they want their ETP to continue to function with later releases. However, this data does not address the possibility that developers who make use of internal APIs are more committed to creating the best possible product.

### 11.11.3 Regression modeling

Survival data implicitly contains information that is not present in other forms of regression modeling: the probability of an event occurring at a given time, i.e., a hazard function. Estimating the appropriate hazard function for survival data requires knowing the coefficients of the explanatory variables in the regression model, while estimating the coefficients of the explanatory variables requires knowing the hazard function.

When building a model, R functions will attempt to fit the shape of the hazard function specified (by the analyst), but if this hazard function is incorrect, the returned model may be substantially incorrect. In practice, parametric models have been found to be very sensitive to the explanatory variables provided as input to the model fitting process.

There is no single statistic available for definitively selecting the best model (i.e., hazard function and appropriate explanatory variables).

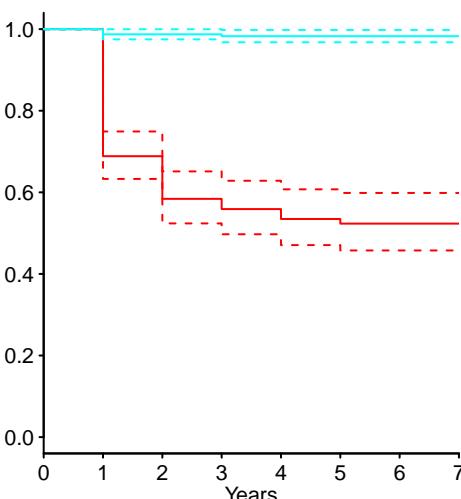


Figure 11.73: The Kaplan-Meier curve for survivability of ETPs ability to be built using SDK released in subsequent years: (blue) ETPs using only official APIs, (red) ETPs calling internal APIs; dotted lines are 95% confidence intervals. Data from Businge.<sup>235</sup> [code](#)

The Cox proportional-hazards model does not require the specification of a hazard function, which breaks the circularity of needing to select regression coefficients for such a function and removes some of the dangers associated with use of an incorrect hazard function (the Cox modeling approach is not guaranteed to always build a reasonably accurate model). If there is any doubt about the appropriate parametric distribution, the Cox model is a safe choice.

While the Cox proportional hazards model has many advantages, a potentially big disadvantage is that without specifying a hazard function, it is not possible to make predictions outside of the interval covered by the measurements.

The `flexsurv` package supports the fitting of complex parametric distributions, and the `cenReg` package supports fitting regression models to censored data.

### 11.11.3.1 Cox proportional-hazards model

The Cox proportional-hazards regression model has been found to provide reasonably good estimates for the coefficients of the explanatory variables and hazard ratios (not absolute values, ratios) for a wide variety of data. The Cox model is popular because it is robust, and will closely approximate the correct parametric model. If the correct parametric model has a Weibull hazard function (whose shape parameter is unknown), the Cox model will give similar results to those obtained from this parametric model. If the parameters of the Weibull hazard function are known, a model built using them will outperform a Cox model.

The Cox likelihood (known as a *partial likelihood*) is based on the observed order of events, rather than the interval between them (so it only considers subjects' experiencing an event).

In the equation for the basic Cox model, time is not included as an explanatory variable,  $x_{ki}$ , i.e., the variables cannot be time dependent. The equation is:

$$h_i(t) = h_0(t)e^{\beta_1 x_{1i} + \dots + \beta_k x_{ki}}$$

where:  $h_i(t)$  is the hazard function for subject  $i$  at time  $t$ ,  $h_0(t)$  is a baseline hazard function, the contents of the exponent expression are explanatory variables and their regression coefficients ( $\beta_0$  is included as part of the baseline hazard).

This equation can be written as a log ratio of the hazard functions:

$$\log \frac{h_i(t)}{h_0(t)} = \beta_1 x_{1i} + \dots + \beta_k x_{ki}$$

or, as a hazard ratio for two subjects,  $i$  and  $j$  (where,  $h_0(t)$ , the baseline hazard function cancels out):

$$\frac{h_i(t)}{h_j(t)} = e^{\beta_1(x_{1i}-x_{1j}) + \dots + \beta_k(x_{ki}-x_{kj})}$$

In this proportional hazards model, the effect of each explanatory variable is multiplicative on the hazard function. In accelerated failure time (AFT) models the multiplicative effect is on the survival function.

The `coxph` function, in the `survival` package, builds Cox proportional-hazard models; the basic usage follows the pattern used by `glm`, with the object returned by `Surv` playing the role of the response variable. For example:

```
p_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource,
            data=ISR_disc)
```

The `cox.zph` function can be used to check the assumption that the explanatory variables are not time dependent (at least during the measurement period).

If two or more events occur at the same time the associated data is said to be *tied*. The default value of the option `ties="efron"`, can handle some tied data, but if many events occur at the same time (e.g., the ETP data in Table 11.6), calls to `coxph` might need to use `ties="exact"`.

The techniques for formula specification and refinement used with `glm` can also be applied to models created with `coxph`, e.g., starting with a complicated model and using `stepAIC` to simplify it.

A study by Arora, Krishnan, Telang and Yang<sup>59</sup> investigated the time taken by vendors to release patches, to fix vulnerabilities reported in their product; explanatory variables included

information about the software vendor, whether the vendor was privately notified about the vulnerability or the vendor first found out about it through a public disclosure.

The following is the summary output from a model fitted by `coxph` to the data for public disclosure vulnerabilities: `code`

```
Call:
coxph(formula = Surv(patch_days, !is_censored) ~ log(cvss_score) +
    opensource + y2003 + smallvendor + small_loge + log(cvss_score):y2002 +
    y2002:smallvendor + y2003:smallvendor, data = ISR_np)

n= 945, number of events= 824

              coef exp(coef) se(coef)      z Pr(>|z|)
log(cvss_score) 0.23283  1.26217  0.08570  2.717  0.00659 ***
opensource       0.42235  1.52555  0.09167  4.607 4.08e-06 ***
y2003            0.83643  2.30811  0.10459  7.997 1.27e-15 ***
smallvendor     -0.40940  0.66405  0.17331 -2.362  0.01816 *
small_loge        0.02926  1.02969  0.01346  2.173  0.02975 *
log(cvss_score):y2002 0.23048  1.25920  0.04961  4.646 3.39e-06 ***
smallvendor:y2002  0.59685  1.81638  0.19540  3.054  0.00226 **
y2003:smallvendor 0.58999  1.80396  0.22502  2.622  0.00874 **

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

              exp(coef) exp(-coef) lower .95 upper .95
log(cvss_score)      1.262      0.7923   1.0670   1.4930
opensource          1.526      0.6555   1.2747   1.8258
y2003               2.308      0.4333   1.8803   2.8332
smallvendor         0.664      1.5059   0.4728   0.9326
small_loge          1.030      0.9712   1.0029   1.0572
log(cvss_score):y2002 1.259      0.7942   1.1425   1.3878
smallvendor:y2002   1.816      0.5505   1.2384   2.6640
y2003:smallvendor   1.804      0.5543   1.1606   2.8039

Concordance= 0.647 (se = 0.012 )
Rsquare= 0.19 (max possible= 1 )
Likelihood ratio test= 199 on 8 df,  p=<2e-16
Wald test           = 184.9 on 8 df,  p=<2e-16
Score (logrank) test = 198.3 on 8 df,  p=<2e-16
```

The first half of the output is similar to the summary output produced by a model fitted using `glm`. The table of numbers in the middle are 95% confidence intervals, which are printed by default. The bottom section of the output lists the R-squared of the fit<sup>xxviii</sup> (0.19 in this case, showing that only a small amount of the variance in the data is described by the model) and p-values for various tests of the null hypothesis that the coefficients are zero (abbreviated to a single letter, p).

The explanatory variable coefficients are proportions, not absolute values (the Cox model is a proportional-hazards model). The coefficients specify the expected impact of the respective explanatory variable, when the values of all the other variables are kept constant. The explanatory variables cannot be used to independently calculate response variable values, they can only be used to predict the change in a known value of the response variable (i.e., the value of the response variable known to occur for specific values of the explanatory variables).

Taking the values in the `log(cvss_score)` row as an example, the value 1.26217 appears in its `exp(coef)` column; what impact will a  $\pm 1$  change in the value of `log(cvss_score)` have on the response variable (i.e., time taken to produce a patch)? The percentage change in the response variable is:  $\pm(1.26217 - 1) \times 100 \rightarrow \pm26.21\%$ ; a value of less than one, in the `exp(coef)` column, reverses the sign of the percentage change, e.g., an increase in the value of the explanatory variable is predicted to decrease the value of the response variable.

Model adequacy can be checked using Cox-Snell residuals, and influential observations checked using `score residuals` (which specify how each regression coefficient would change if a particular observation was removed; see [survival/vulnerabilities/patch-cph.R](#)).

<sup>xxviii</sup> The value printed is the Cox & Snell pseudo R-squared, which can be less than one; the maximum possible value for the data is given in the `summary` output.

**Frailty of subjects** Unobserved differences in subject performance may result in some variation in the hazard function they experience,<sup>70</sup> *frailty* is the term used to denote these random (multiplicative) changes in hazard function. Introducing a random effect,  $v_j$ , the frailty of group  $j$  that  $x_i$  belongs to, modifies the Cox model as follows:

$$h_i(t) = h_0(t)v_j e^{\beta_1 x_{1i} + \dots + \beta_k x_{ki}}$$

The previous analysis of time-to-patch, implicitly assumes there is no difference between vendors, in their ability to respond and fix reported vulnerabilities. The *frailty* function can be included in a formula, to specify explanatory variables that identify particular groups of subjects sharing the same frailty.

```
fp_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource
+frailty(vendor), data=ISR_disc)
```

The summary output includes the information: Variance of random effect=0.374 (see [survival/vulnerabilities/patch-frailty.R](#)).

The  $v_j$  in the above equation is assumed to have a mean=1 and a variance that is calculated as part of the model building process (in this case it is 0.374). The main consequence of including frailty in a Cox model is to explicitly allocate some of the variance present in the data to a specific explanatory variable (the model coefficients of explanatory variables may also change).

The *frailtypack* package provides a wider range of frailty related options and functionality, than is available in the *survival* package. The *coxme* package supports the fitting of mixed-effects Cox models (frailty can be handled as a specific kind of random effect).

### 11.11.3.2 Time varying explanatory variables

The behavior of an explanatory variable may change over time. Options for handling this behavior includes, excluding all affected subjects from the analysis or using a technique that handles time dependent behavior

The Arora et al study (discussed earlier) investigated the impact of public disclosure of vulnerabilities, on the time taken by vendors to release patches for their product. Possible event sequences were:

- vendor was privately notified about a vulnerability and some time later a simultaneous announcement of the vulnerability and a vendor patch was made (213 of 755 private notifications),
- vendor was privately notified about a vulnerability, but information about the vulnerability was later made public before a patch was available for release (the vendor's patch being released some time later in 542 of 755 private notifications); this is a time dependent change of a significant attribute.
- the vendor learned about a vulnerability when information about it was made public, and sometime later released a patch (945 cases),

If privately notified and public disclosure fix rates are compared using a Kaplan-Meier curve, any privately notified vulnerabilities that become public before a patch is available have to be treated as censored (ignoring them biases fix rates towards a lower value; see Figure 11.74).

The Cox models discussed earlier, were fitted using vulnerability data where the vendor found out about the vulnerability via public disclosure.

Building a regression model using all the vulnerability data, requires handling time dependent explanatory variables; the data has to be restructured to make the time dependencies explicit. The time dependency, for this data, is a possible change of state, from the vulnerability not being public, to the information being public.

The original data looks something like the following:

```
notify,publish,patch,vendor,employee,os
2000-10-16,2000-11-18,2000-12-20,"abc",1000,unix
```

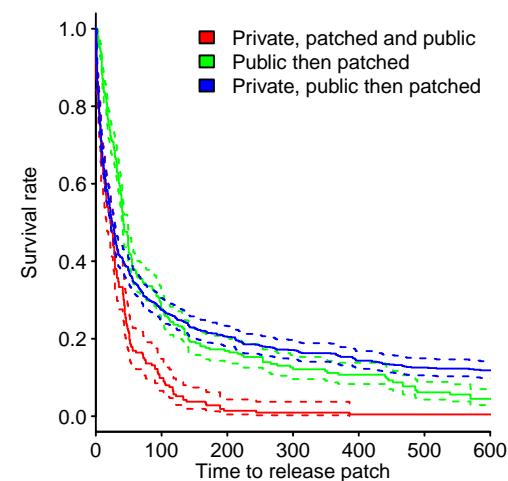


Figure 11.74: Kaplan-Meier curves for time-to-release a patch for a reported vulnerability, with private, public and private then public notification. Data from Arora et al.<sup>59</sup> [code](#)

When vulnerability information is made public before a patch is released, extra information is involved. For this data, five columns are added: one to uniquely identify each vulnerability, the start/end dates of the interval during which the information was private or disclosed, a flag specifying private/disclosed, and a flag for whether an event (i.e., release of a patch) occurred in the interval. The first interval starts on the date the vendor was notified and ends on the date the vulnerability is made public, a second interval occurs for vulnerabilities that change state from private to disclosed before a patch is available and starts on the date of disclosure and ends on the date a patch became available, as follows:

```
id,start,end,priv_di,notify,publish,patch,event,vendor,os
1,2000-10-16,2000-11-17,1,2000-10-16,2000-11-18,2000-12-20,0,"abc",unix
1,2000-11-18,2000-12-20,0,2000-10-16,2000-11-18,2000-12-20,1,"abc",unix
```

Treating `priv_di` as an explanatory variable (1 for private disclosure to vendor and zero for public disclosure), enables the impact of disclosure on patch time to be included in a model.

When all the measurement data needs to be split on the same date, the `survSplit` function can be used to create the necessary rows, otherwise (as in this case) specific data mangling code has to be written.

The call to `coxph`, or `survreg`, has to include the term `cluster(id)`, which ties together (by vulnerability id in this case) the rows associated with the same subject. The call to `coxph` looks something like the following:

```
td_mod=coxph(Surv(patch_days, !is_censored) ~ priv_di*cvss_score
              +cluster(id), data=ISR_split)
```

It is not possible for both `cluster` and `frailty` to appear in the same formula (`cluster` is based on GEE model building, while `frailty` is based on mixed-effects model building).

The `summary` output for the time dependent model is: [code](#)

Call:

```
coxph(formula = Surv(patch_days, !is_censored) ~ cluster(ID) +
      priv_di * (cvss_score + c_o + dis_by_s + os + y2 + smallvendor +
      small_loge) - c_o - dis_by_s - os - smallvendor + cvss_score:(c_o +
      dis_by_s + s_app) + opensource:c_o + opensource:dis_by_s +
      os:s_app + s_app:y2, data = ISR_split)
```

n= 2242, number of events= 2081

	coef	exp(coef)	se(coef)	robust	se	z
priv_di	2.798750	16.424106	0.216150	0.209360	13.368	
cvss_score	0.153926	1.166404	0.016806	0.017733	8.680	
y2	0.277421	1.319722	0.044042	0.044590	6.222	
small_loge	0.037114	1.037811	0.007262	0.008817	4.210	
priv_di:cvss_score	-0.114788	0.891555	0.017327	0.016795	-6.835	
priv_di:c_o	0.644347	1.904743	0.228463	0.211989	3.040	
priv_di:dis_by_s	0.475405	1.608665	0.116261	0.106601	4.460	
priv_di:os	-0.331847	0.717597	0.098936	0.086976	-3.815	
priv_di:y2	-0.614162	0.541094	0.063296	0.061954	-9.913	
priv_di:smallvendor	-0.440845	0.643492	0.138900	0.099310	-4.439	
priv_di:small_loge	-0.082120	0.921161	0.016589	0.014449	-5.683	
cvss_score:c_o	-0.060084	0.941685	0.012861	0.011990	-5.011	
cvss_score:dis_by_s	-0.061114	0.940716	0.008798	0.011002	-5.555	
cvss_score:s_app	-0.096771	0.907764	0.014972	0.014853	-6.515	
c_o:opensource	0.443978	1.558896	0.137952	0.118459	3.748	
dis_by_s:opensource	0.414151	1.513086	0.091161	0.102359	4.046	
os:s_app	0.815803	2.260991	0.077450	0.093536	8.722	
y2:s_app	0.291007	1.337774	0.047420	0.045599	6.382	
	Pr(> z )					
priv_di	< 2e-16	***				
cvss_score	< 2e-16	***				
y2	4.92e-10	***				
small_loge	2.56e-05	***				
priv_di:cvss_score	8.22e-12	***				
priv_di:c_o	0.002369	**				
priv_di:dis_by_s	8.21e-06	***				
priv_di:os	0.000136	***				

```

priv_di:y2           < 2e-16 ***
priv_di:smallvendor 9.03e-06 ***
priv_di:small_loge   1.32e-08 ***
cvss_score:c_o      5.42e-07 ***
cvss_score:dis_by_s 2.78e-08 ***
cvss_score:s_app    7.27e-11 ***
c_o:opensource      0.000178 ***
dis_by_s:opensource 5.21e-05 ***
os:s_app            < 2e-16 ***
y2:s_app            1.75e-10 ***

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Concordance= 0.654  (se = 0.007 )
Rsquare= 0.231  (max possible= 1 )
Likelihood ratio test= 590.1  on 18 df,  p=<2e-16
Wald test            = 376.9  on 18 df,  p=<2e-16
Score (logrank) test = 586.8  on 18 df,  p=<2e-16,  Robust = 454.6  p=<2e-16

(Note: the likelihood ratio and score tests assume independence of
observations within a cluster, the Wald and robust score tests do not).

```

There are two parts to the contribution made by `priv_di`; as a standalone variable it has a large impact, but its interactions with other variables create a large impact in the opposite direction (the model building process tries to minimise its error metric, not make it easy for analysts to understand what is going on).

The component of the fitted equation of interest is:

$$e^{priv\_di(2.8 - 0.11cvvs\_scorei + 0.64c\_o + 0.48dis\_by\_s - 0.33os - 0.61y2 - 0.44smallvendor - 0.08small\_loge)}$$

where: `priv_di` is 0/1, `cvvs_score` varies between 1.9 and 10 (mean 7), `c_o` 0/1 NA other<sup>xxix</sup> (mean 0.13), `dis_by_s` 0/1 disclosed by SecurityFocus (mean 0.38), `os` 0/1 vulnerability in O/S (mean 0.26), `y2` years since 2000 (mean 1.9), `smallvendor` 0/1 small vendor flag (mean 0.25) and `small_loge` zero for small vendors, otherwise the log of number of employees (mean 5).

Applying some hand waving to average away variables:

$$e^{priv\_di \times (2.8 - 0.11 \cdot 7 + 0.64 \cdot 0.13 + 0.48 \cdot 0.38 - 0.33 \cdot 0.26 - 0.61 \cdot 1.9 - 0.44 \cdot 0.25 - 0.08 \cdot 5)}$$

$$\rightarrow e^{priv\_di \times (2.8 - 0.77 + 0.08 + 0.18 - 0.09 - 1.2 - 0.11 - 0.4)} \rightarrow e^{0.49priv\_di}$$

to produce a (hand waving mean) percentage increase of  $(e^{0.49} - 1) \times 100 \rightarrow 63\%$ , when `priv_di` changes from zero to one. The percentage change for patches, for vulnerabilities with a low `cvvs_score` is around 90% and for a high `cvvs_score` around 13% (i.e., the patch time of vulnerabilities assigned a low priority improves a lot when they are publically disclosed, but patch time for those assigned a high priority is only slightly affected).

The process of calculating the 95% confidence bounds, based on the values in the `summary` output, is fiddly and left to the reader.

Time varying changes may occur, but the information needed to model these changes may not be available.

A study by Lunesu<sup>988</sup> investigated the maintenance activities of a large software company; between 2005 and 2010 there were 5,854 issues. The response time for an issue (i.e., the time between an issue being opened to it being closed) depends on the rate issues are reported and the resources available to handle issues.

Extra resources were added to handle the growing number of issues (after around 400 days) and the number of issues reported decreased after around 800 days (it is not known whether this resulted in issue handling resources being decreased, or the same level of resources applied to fewer issues). The level of resources used to resolve issues cannot be included in a model, because this information is not available.

Figure 11.75 shows cumulative number of issues reported and closed, over time (upper); the lower plot shows the survival curve for issues reported in the first 400 days, reported between 400 and 800 days and reported after 800 days. As expected, the extra resources added after

<sup>xxix</sup> No information is available on what this variable represents.

400 days reduced open issue survival times, but the reduction in reported issues after 800 days does not appear to have had much impact on survival rates (perhaps because it is easier to move existing staff to other work, than to add new staff).

#### 11.11.4 Competing risks

When more than one possible kind of event can occur (i.e., there are multiple terminal states), a competing risk model might be used or each distinct event type might be analyzed separately from the other event types (data involving other events is flagged as censored at the time the other event occurs).

The Kaplan-Meier plot for a single event, in a competing risk context, may give a misleading impression of the actual situation for events that rarely occur. The *cumulative incidence curve* (CIC) is a commonly used alternative, which includes information on every event (when there is only one event,  $CIC = 1 - KM$ ). CIC does not assume that competing risks are independent and estimates the marginal probability of an event.

The `cmprsk` package supports the modeling of competing risks.

A study by Di Penta, Cerulo and Aversano<sup>400</sup> investigated the history of mistakes in source code flagged by various static analysis tools. Newly written source code containing a flagged construct was tracked through subsequent versions. Possible competing events include the removal of the code containing the flagged construct and the flagged construct being modified such that it is no longer flagged (e.g., a bug fix).

Figure 11.76 shows the cumulative incidence curves (created by the `cuminc` function, in the `cmprsk` package) for problems reported in the Samba and Squid source by the splint static analysis tool.

```
library("cmprsk")
plot_cif=function(sys_str)
{
  t=cuminc(rats$faitime, rats$type, cencode=0, subset=(rats$SYSTEM == sys_str))
  plot(t, col=pal_col, cex=1.25,
        curvlab=c("was removed", "disappeared"),
        xlab="Snapshot", ylab="Proportion flagged issues 'dead'\n")
  text(max(t[[1]]$time)/1.5, 0.9, sys_str, cex=1.5)
}

plot_cif("samba")
plot_cif("squid")
```

#### 11.11.5 Multi-state models

Multistate models deal with time to event processes, where there are multiple events with potential changes of state between them.

The `mstate` package supports the building of multi-state Cox models and competing risk models, the `msm` package supports the building of multi-state Markov models.

A study by Goeminne and Mens<sup>581</sup> investigated the evolution in the use of four database frameworks in 2,818 Java projects. A project might start using, say, Spring, then add support for another framework or switch from using Spring to a different framework.

What is the probability of changes, over time, in the number of database frameworks, used by a project?

The following call to `msm` fits a multi-state Markov model for the number of database frameworks (`dbs`) used by projects (`X`) over time (`date`). The matrix `Q` is an initial estimate of the state transition probabilities of a project currently using  $i$  frameworks migrating to use  $j$  frameworks; any transition that cannot occur must contain zero in the corresponding non-diagonal element (specifying `gen.inits=TRUE` results in an approximate estimate being calculated internally; see [survival/icsme2015era.R](#)).

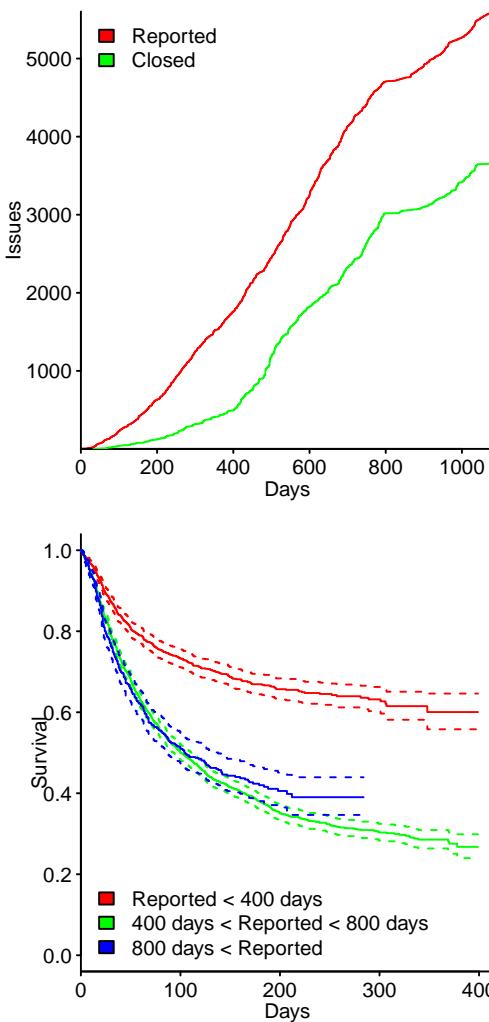


Figure 11.75: Cumulative number of issues reported and closed, and issue survival curves for three intervals. Data from Lunesu.<sup>988</sup> [code](#)

```

library(msm)

Q=matrix(nrow=4, ncol=4,
         c(0, 0.1, 0.1, 0.01,
           0.1, 0, 0.1, 0.01,
           0.1, 0.1, 0, 0.1,
           0.1, 0.1, 0.1, 0))

db msm=msm(dbs ~ date, subject=X, data=uses,
            qmatrix=Q, gen.inits=TRUE, exacttimes=TRUE)

pmatrix.msm(db msm, t=365)
sojourn.msm(db msm)

```

Table 11.7 shows the estimated likelihood of a project migrating from using  $i$  database frameworks to using  $j$  frameworks, within 365 days.

from	to 1	to 2	to 3	to 4
1	0.89	0.09	0.02	0.00
2	0.07	0.74	0.17	0.03
3	0.02	0.07	0.77	0.14
4	0.01	0.01	0.05	0.93

Table 11.7: Estimated likelihood that within 365 days, a project using  $i$  database frameworks will migrate to using  $j$  frameworks. Data from Goeminne et al.<sup>581</sup>

Some of the estimation functions supported by the `msm` package include: the `pmatrix.msm` function estimates the state transition probabilities, within a given time  $t$ , and the `sojourn.msm` function which estimates the mean time spent in each state.

## 11.12 Circular statistics

Some measurements are based on a circular scale, with values wrapping around to the minimum value when incremented past the maximum value, e.g., time of day or months of the year. Circular statistics<sup>1241</sup> is the name given to the analysis of data measured using such a scale. Circular statistics has only become widely studied in the last 40 years or so, and techniques for handling operations that are well-established in other areas of statistics are still evolving. The `circular` package supports the analysis of data measured using a circular scale.

Differences between measurements based on circular and linear scales include:

- plotting uses a polar representation, rather than x/y-axis (the `circular` package includes support for `plot`, `lines`, `points` and `curve` functions),
- the mean has two components: mean direction ( $\bar{\theta}$ , an angle) and mean resultant length ( $\bar{R}$ ; if this value is zero, the data has no mean), returned by the `mean` and `rho.circular` functions respectively (the `trigonometric.moment` function provides another way of obtaining this information). The `median.circular` function returns a median (multiple medians may exist, but only one is returned),
- the term variance, on its own, is ambiguous. The *circular variance*,  $V$ , is defined as  $V = 1 - \bar{R}$  and varies between zero and one. Another measure is *angular variance* (returned by the `angular.variance` function), which varies between zero and two.

The *circular standard deviation* is returned by the `sd.circular` function (it is not calculated by taking the square root of the variance; its formula is:  $\sqrt{-2 \log \bar{R}}$ ),

- the von Mises distribution plays a role similar to that filled by the Normal distribution on linear measurement scales.

The mean resultant length,  $\bar{R}$ , is a measure of how spread out data points are around the circle. If the points have a symmetric distribution  $\bar{R}$  equals zero and if all the points are concentrated

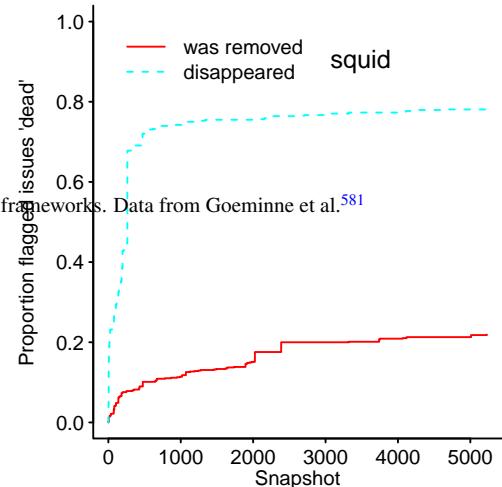
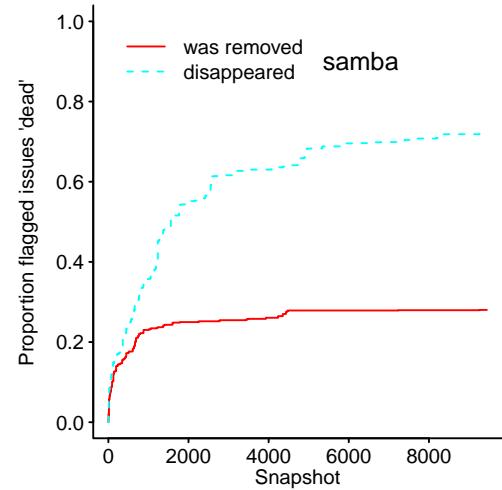


Figure 11.76: Cumulative incidence curves for problems reported by the splint tool in Samba and Squid (time is measured in number of snapshot releases). Data from Di Penta et al.<sup>400</sup> code

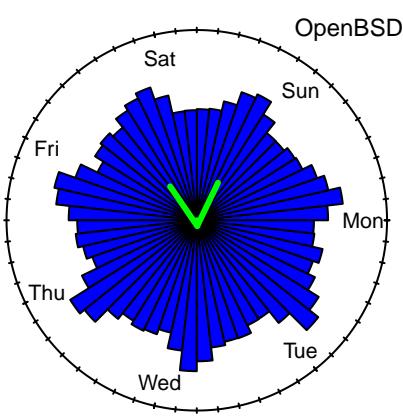
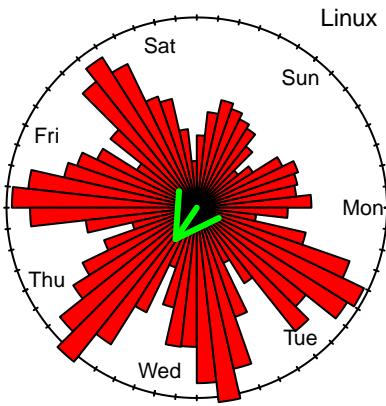


Figure 11.77: Rose diagram of number of commits in each 3 hour period of a day for Linux and FreeBSD. Data from Eyalson et al.<sup>462</sup> [code](#)

in one direction  $\bar{R}$  equals one; for unimodal distributions the term *concentration* is applied to  $\bar{R}$  to denote the extent to which measurements concentrate around the mean direction.

Figure 11.77 is a Rose diagram of the number of commits to Linux and FreeBSD for each 3 hour period of the days of the week (the same data is plotted using a linear scale in Figure 5.6).

The `rose.diag` function plots Rose diagrams. By default, the area of each segment is proportional to the number of measurement points in the segment (the behavior used when plotting histograms).

```
library("circular")

# Map to a 360 degree circle
HoW=circular((360/hrs_per_week)*week_hr, units="degrees", rotation="clock")
rose.diag(HoW, bins=7*8, shrink=1.2, prop=5, axes=FALSE, col=col_str)
axis.circular(at=circular(day_angle, units="degrees", rotation="clock"),
              labels=c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"))

text(0.8, 1, repo_str, cex=1.4)
arrows.circular(mean(HoW), y=rho.circular(HoW), col=pal.col[2], lwd=3)
```

The arrow at the center shows the direction of the mean and the length of its shaft is its resultant length. Linux has fewer commits at weekends, compared to weekdays and a mean direction near the middle of the week looks reasonable. The number of commits to FreeBSD does not seem to vary between days; the mean length is 0.03 (it almost does not have a mean), compared to Linux's mean length of 0.2.

If the measurement scale is very course (e.g., measuring commit time to an accuracy of a day, rather than an hour or minute), then  $\bar{R}$  will be underestimated and introduce errors in the calculation of the various location measures which use this value (see [statistics/circular/circle-bin.R](#)). The suggested correction to  $\bar{R}$ , for calculating circular standard deviation, when measuring in units of days rather minutes, is to multiply the calculated value of  $\bar{R}$  by 1.034 (the correction for higher order moments involves much larger values).

### 11.12.1 Circular distributions

Circular distributions that are often encountered include the uniform distribution, wrapped Cauchy, wrapped von Mises and the Cartwright distributions.

Figure 11.78 shows differences in the shapes of three popular, symmetrical, single peak, wrapped circular distributions.<sup>xxx</sup> The *Jones-Pewsey distribution* includes them all, and others, as special cases.

Figure 11.79 shows asymmetric extended forms of some common circular distributions. The `circular` package does not include support for asymmetric distributions, but code is available in Pewsey et al.<sup>1241</sup>

The *discrete circular uniform distribution* consists of  $m$  points, equally spaced around a unit circle, with each point occurring with probability  $\frac{1}{m}$ .

The *continuous circular uniform distribution* treats all directions as being equally likely; the probability of point occurring between the angles  $\phi$  and  $\psi$  is:  $P(\phi < \theta < \psi) = \frac{\psi - \phi}{2\pi}$ . The `circular` package supports the `dcircularuniform` and `rcircularuniform` functions, but not `p` and `q` forms.

The choice of which circular uniformity test to use, for measurements on a continuous scale, i.e., many possible measurement points around the circle, depends on how the data is thought to deviate from uniformity. The two uniformity deviation possibilities are:

- a single peak over some range of values, i.e., a unimodal distribution. In this case the Rayleigh test is the most powerful known test; available in the `rayleigh.test` function,
- multiple peaks in the distribution of values around the circle. There are three tests which are more powerful than the Rayleigh test, when the data distribution could be more complicated than a single peak, but no single one is superior to the others; support for these tests is available in the `kuiper.test`, `watson.test` and `rao.spacing.test` functions.

<sup>xxx</sup> The implementation of the Cartwright distribution, up to at least version 0.4.93 of the `circular` package, uses the spelling `carthwrite`.

Unless there is a good reason to think that the measurements could have a single peak, one (or all) of the omnibus tests should be used.

Your author was once a member of a four person compiler implementation team, all born in February; we all agreed that the best compiler implementers are born in February. An alternative, easier to test hypothesis, is that most compiler implementers are born in February. Your author ran a survey on his compiler oriented blog,<sup>790</sup> asking readers their birth month and whether they had spent more than four months working on a compiler project (132 responded, of which 82 had worked on a compiler).

Figure 11.80 shows that while February was the most common birth month, with 15% of implementers it is substantially below a majority (weighting by the number of days in a month, or the yearly percentage of births in each month, does not have much impact).

How likely is it that compiler implementer birthdays are uniformly distributed over the months? When the measurements have been grouped into a few bins, e.g., months of the year, a grouped data test has to be used; see `statistics/circular/grp-data-boot.R` for examples using bootstrap. All tests for uniformity fail.

A study by Eyolfson, Tan and Lam<sup>463</sup> investigated the correlation between commit time and the likelihood of a fault being experienced because of a mistake in the commit code, for Linux and PostgreSQL. Figure 11.81 shows the number of non-fault commits (upper) and number of commits in which a fault was detected (lower), made in each hour of combined weekdays (the pattern of commits on weekdays differs from weekend days and the following analysis is based on weekdays only).

What differences, if any, exist between the two sets of daily commit times and in particular are commits made at certain times of the day more likely to cause a fault experience?

- testing for a common mean direction: The `watson.williams.test` function assumes that both samples are drawn from a von Mises distribution; the *Watson large sample non-parametric test* does not even require the samples to share a common shape (see `statistics/circular/common-mean.R`). When any of the samples has a size less than 25, a bootstrap version of these tests should be used. The daily commit times do not share a common mean direction (15.5 hours for fault commits and 16.2 hours for non-fault commits); the mean result lengths are 0.33 and 0.32 respectively,
- testing for a common concentration: Are the points concentrated around a common direction? The *Wallraff test* is not supported by the `circular` package, but is described in Pewsey et al<sup>1241</sup> (see `statistics/circular/common-concen.R`). The two commit samples do not share a common concentration.

## 11.12.2 Fitting a regression model

When one or more variables are measured on a circular scale the technique used to build a regression model depends on whether the circular variable is an explanatory or response variable.

When the response variable is measured on a linear scale, existing techniques and functions can be used; there may be one or more circular or linear explanatory variables,

When the response variable is measured on a circular scale, the `lm.circular` function, in the `circular` package, can be used

### 11.12.2.1 Linear response with a circular explanatory variable

Circular explanatory variables can be modeled using periodic functions and the regression modeling techniques discussed in earlier sections; sine and cosine functions can be combined to model any periodic function. As always, a model containing the fewest number of distinct parameters is desired.

The cosine function can be modified in various ways to change its shape, including:

$$y = \alpha + \beta \cos(\omega x + \phi)$$

and higher order harmonics can be added:

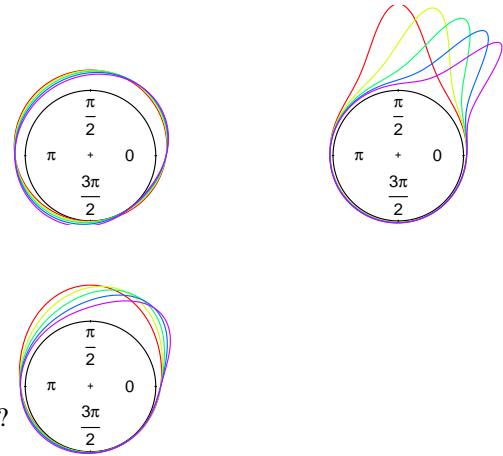


Figure 11.79: Asymmetric extended wrapped forms of the Cardioid (upper), von Mises (middle) and Cauchy (lower) probability distributions for various values of their parameters. [code](#)

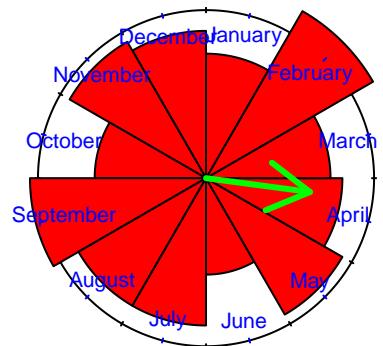


Figure 11.80: Number of readers of author's blog, whose birthday falls within a given month and who have worked on a compiler. Data from Jones.<sup>790</sup> [code](#)

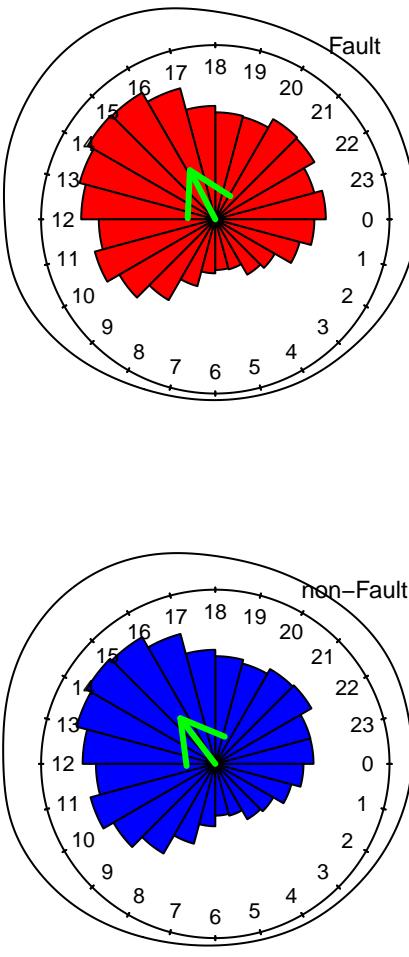


Figure 11.81: Number of commits (upper) and number of commits in which a fault was detected (lower) by hour of day of the commit, for Linux. Data from Eyolfson et al.<sup>463</sup>  
code

$$y = \alpha + \beta_1 \cos(\omega x + \phi) + \beta_2 \cos(2\omega x + \phi) \dots$$

The shape of the peaks and troughs can be modified by adding a sine wave to the angular argument. In the following a positive  $\lambda$  sharpens the peaks and flattens the troughs while a negative  $\lambda$  has the opposite effect.

$$y = \alpha + \beta \cos(\omega x + \phi + \lambda \sin(\omega x + \phi))$$

A skewed period (which is what asymmetrical distributions have) can be modeled by adding a cosine wave to the angular argument (provided  $-\pi/6 \leq \lambda \leq \pi/6$ ; outside this range it also effects other shape characteristics):

$$y = \alpha + \beta \cos(\omega x + \phi + \lambda \cos(\omega x + \phi))$$

These are all non-linear equations and the `nls` function can be used to fit them.

The commit data, seen in Figure 11.81, is asymmetrical and the following code fits an extended cosine regression model (the `gam` values were estimated from the height of the cycle and `omega` from fitting 24 hours into  $2\pi$  radians):

```
basic_mod = nls(freq ~ gam0+gam1*cos(omega*hour-phi)+nu*cos(omega*hour-phi),
                 start=list(gam0=800, gam1=700, omega=0.3, phi=1, nu=0),
                 data=week_basic)
```

The upper plot in Figure 11.82 shows the number of non-fault commits per hour for every week day and the fitted model, the lower plot shows the commits with detected faults.

Both fits handle the skewed period but not the sharp peak and flat trough. A sine contribution can be added to help handle this shape and improve the fit, the call to `nls` is below:

```
basic_2mod = nls(freq ~ gam0
                  +gam1*cos(omega*hour-phi+nu*cos(omega*hour-phi))
                  +gam2*cos(2*omega*hour-phi+nu*sin(omega*hour-phi)),
                  start=list(gam0=800, gam1=700, gam2=100,
                             omega=0.3, phi=1, nu=0),
                  data=week_basic)
```

Figure 11.83 overlays the fitted curve for non-fault and fault (red) commits over the non-fault hourly commits for each workday.

Confidence intervals...

### 11.12.2.2 Circular response variable

The `lm.circular` function supports circular response variables... if only we had some data...

## 11.13 Compositions

Compositional data is made up of components whose total contributions sum to 100 (or 1). The requirement of a fixed total creates a correlation between the components, i.e., if one of them increases, one or more of the others has to decrease correspondingly. Failure to take this mutual correlation between variables into account, can result in models having surprising characteristics being fitted to the data.

The theory needed to underpin techniques for handling compositional data became available at the start of the century; it is all very new and many issues are still unsolved.

The `compositions` package supports the analysis of compositional data and offers four approaches to the analysis of amounts (based on the geometries of the sample spaces). The compositional mapping functions are:

- `aplus`: the total amount matters, but amounts are compared relatively, e.g., the difference between 1 and 2 is treated as the same as the difference between 100 and 200,
- `rplus`: the total amount matters, and amounts are compared absolutely, e.g., the difference between 1 and 2 is treated as the same as the difference between 100 and 101,

- `acomp`: the total amount is constant, but amounts are compared relatively, e.g., the difference between 1 and 2 is treated as the same as the difference between 100 and 200,
- `rcomp`: the total amount is constant, and amounts are compared absolutely, e.g., the difference between 1 and 2 is treated as the same as the difference between 100 and 101.

A study by Machiry, Tahiliani and Naik<sup>1000</sup> measured the performance of two application test generators, by comparing the number of lines of program source code covered by the tests generated by each tool (50 Android apps were tested); human performance was also measured.

The application source lines covered by human and tool generated tests was recorded. The difficulty of creating tests to cover source lines is likely to vary across applications and within different parts of the same application. Normalizing coverage counts, to a percentage of source lines covered, allows performance across different applications to be compared.

Figure 11.84 shows the coverage common to human and Dynodroid written tests decreasing (measured as a percentage of all covered lines) as the number of application source lines increases.

One measure of human vs. tool performance is to compare just those source lines that are covered by tests. What percentage, for each application, is covered by both human and tool generated tests and what percentage uniquely covered by human or tool tests? Figure 11.85 shows this information for the Dynodroid tool (the red tick marks on the axis are measurement points where one of the three components is zero), along with a (poorly fitting green) regression line.

The clustering of points near the Human & Dynodroid vertex shows that tests created by these two generators tend to cover the same source lines. More points are near the Dynodroid axis than the Human axis, suggesting that Dynodroid generated tests cover fewer unique source lines.

Fitting three regression models, one for each coverage percentage, using application source lines as the explanatory variable fails to make use of all the available information, i.e., the relationship between the three percentages. A method of combining the three percentages into a single entity, that can be used as a response variable is required. The *isometric log-ratio transformation*, `ilr`, is one possibility and the `compositions` package supports the `ilr` function.

The `acomp` function normalises the columns passed as an argument using a ratio scale, and returns an object having class `acomp` (named after Aitchison who pointed out the useful mathematical properties that a ratio scale bring to compositional analysis).

The `ilr` function is not currently handled by `glm`, so `lm` has to be used. Understanding the following code requires a lot more background knowledge than is appropriate here; see van den Boogaart and Tolosana-Deldago<sup>1587</sup> for more details.

```
library("compositions")

covered=acomp(dh, parts=c("LOC.covered.exclusively.by.Dyno..D.",
                         "LOC.covered.exclusively.by.Human..H.",
                         "LOC.covered.by.both.Dyno.and.Human..C."))

plot(covered, labels="", col=point_col, mp=NULL)
ternaryAxis(side=0, small=TRUE, aspanel=TRUE,
            Xlab="Dynodroid", Ylab="Human", Zlab="Human & Dynodroid")

dh$l_total_lines=log(dh$Total.App.LOC..T.)

comp_mod=lm(ilr(covered) ~ I(l_total_lines^2), data=dh)

d=ilrInv(coef(comp_mod)[-1, ], orig=covered)
straight(mean(covered), d, col="green")
```

The explanatory variable is total source lines in the application and the red plus signs show predictions for various totals. The quality of the fit is very poor, with potentially many outliers and non-constant variance; other explanatory variables, not present in the data, may enable the building of better fitting models.

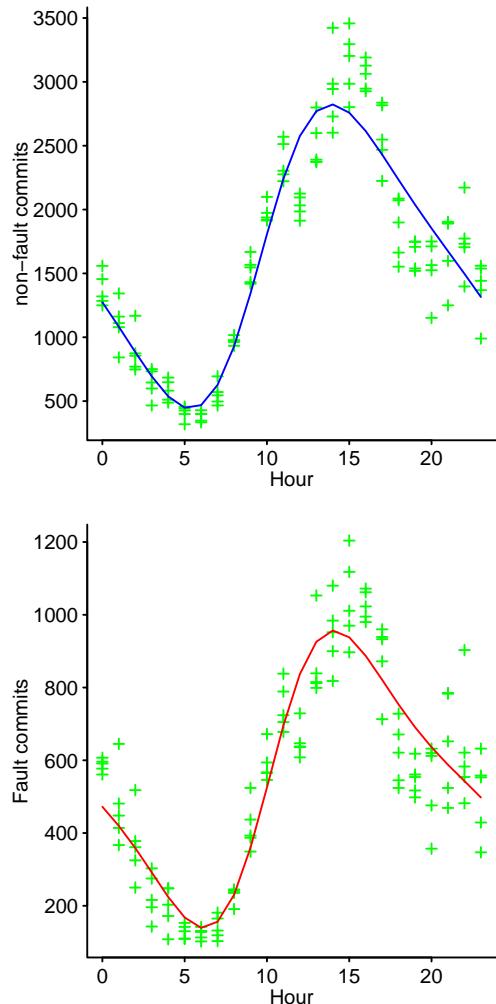


Figure 11.82: Number of commits per hour for weekdays and fitted model (upper) and number of commits in which a fault was detected (lower), for Linux. Data from Eyolfson et al.<sup>463</sup> [code](#)

A study by Hatton<sup>662</sup> investigated 1,294 distinct maintenance activities. For each activity, the developer's estimated and actual percentage time spent on adaptive, corrective and perfective activities was recorded... [regression/Maint\\_Dec2005.R](#)

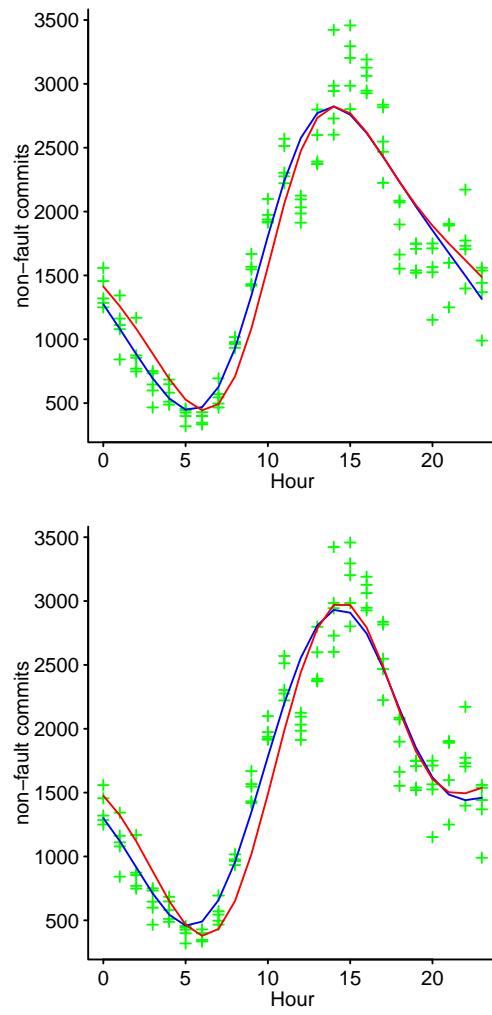


Figure 11.83: Number of commits per hour for each weekday, fitted using  $\cos(\dots\cos\dots)$  (upper) and  $\cos(\dots\cos+\sin\dots)$  (lower), for Linux; in both cases the fitted fault model (red) has been rescaled to allow comparison. Data from Eyolfson et al.<sup>463</sup> [code](#)

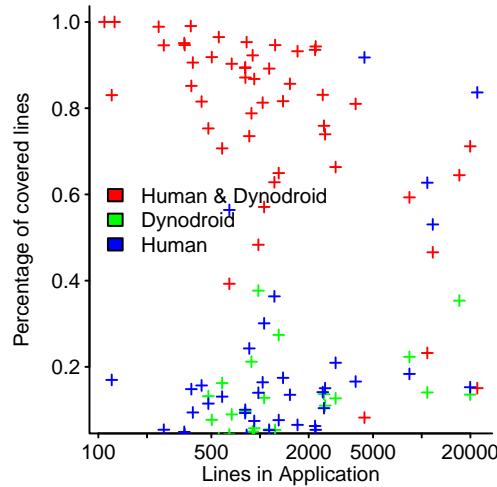


Figure 11.84: Application source lines against percentage of covered lines achieved by both Human & Dynodroid tests, only by Dynodroid tests and only by Human tests. Data from Machiry et al.<sup>1000</sup> [code](#)

# Chapter 12

## Miscellaneous techniques

### 12.1 Introduction

This chapter covers techniques that produce results that do not have the equational form used in regression models.

### 12.2 Machine learning

Machine learning is the name given to a collection of techniques for automatically building a prediction model, based on a collection of training examples.

Users of machine learning do not need to understand the data (although it helps if they do), and as such, this approach to model building is ideal for clueless button pushers. From time to time, we are all clueless button pushers; machine learning is an easier to use tool that can help find a path through the fog.

This book's emphasis is on understanding the processes involved in software engineering, not building black-box prediction models.

The quality of the predictions made by models built using machine learning, depend on the quality of the training data used. It is worth noting that: the blacker the prediction box, the faster feedback is needed on prediction accuracy. Following black box predictions, without regular feedback on their accuracy is a recipe for disaster.

A sample containing information about many variables is always useful to have; domain knowledge might be used to select an appropriate subset. However, when all the variables are included in the analysis at the same time the *curse of dimensionality* arises.

A common metric used by machine learning algorithms is the distance between points. Each measurement can be viewed as a point in an  $n$ -dimensional space, where  $n$  is the number of attributes associated with each measured item. For ease of comparison in the following analysis this  $n$ -dimensional space every side is assumed to have length one, and so its volume is also one. In 3-dimensions the volume of a sphere of diameter one is  $\frac{4}{3}\pi0.5^3 \rightarrow 0.52$ , that is the sphere occupies 52% of the unit cube. If the unit cube contains multiple points, then there is a 52% probability that a point at the center of the unit cube will be within 1-unit distance of another point. As the number of dimensions increases the sphere/unit cube volume ratio increases to a peak at five dimensions, and then decreases rapidly. Figure 12.1 shows how the volume of a sphere changes, relative to the volume of the unit cube, as the number of dimensions increases.

As the number of dimensions increases, the distance from a point to the point nearest to it approaches the distance to the point furthest from it;<sup>156</sup> an effect that can occur for as few as 10-15 dimensions. This behavior means that any algorithm relying on distance between points effectively ceases to work at higher dimensions.

**Text analysis** Software engineering produces often produces large quantities of text written in natural language, e.g., English. Like source code, this natural language text is raw material from which useful information might be extracted.

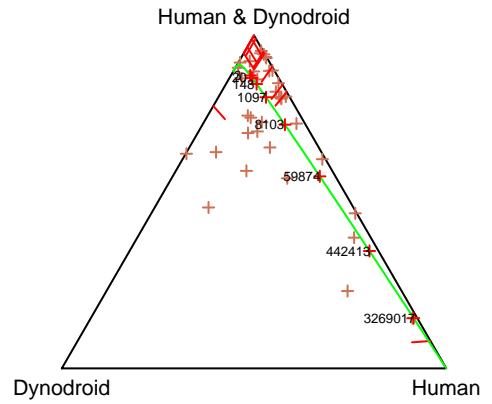


Figure 11.85: Percentage of source lines covered by both Human & Dynodroid tests, by only by Dynodroid tests and only by Human tests; fitted regression line and prediction points for various total source lines, red plus. Data from Machiry et al.<sup>1000</sup> [code](#)

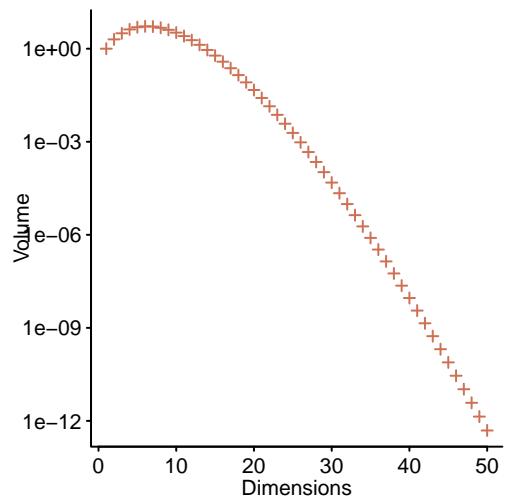


Figure 12.1: Volume of unit sphere in 1 to 50 dimensions, e.g., sphere has volume  $\frac{4}{3}\pi r^3$  in three dimensions. [code](#)

Automated extraction of semantics from natural language is an unsolved problem, for the general case. Approximate answers can sometimes be obtained to specific kinds of semantic questions. Some algorithms are based on using prelearned examples, e.g., sentiment analysis is a popular technique for estimating whether text expresses a positive and negative opinion, but the results depend on the training data and tool used<sup>794</sup> (researchers are starting to collate software engineering specific training data<sup>966</sup>). Dependence on training data is an important issue for any approach based on using pretrained models.

Available R packages for text analysis include `tm` (along with extension packages, such as `tm.plugin.mail` for processing emails and `tm.plugin.webmining` for mining web pages) and `spacyr` provides an interface to the `spacy.io` natural language processing system. For an example, see `faults/reopened_text.R`.

### 12.2.1 Decision trees

As the name suggests decision tree models take the form of a tree like structure, rather than of an equation. Each node of the tree contains either an expression whose result is used to select which of two branches to follow or a value denoting the result. The `rpart`<sup>i</sup> package supports the creation of binary decision trees.

Tree models are popular for use cases where a models is needed that can be interpreted by the people making a decision based on what they observe, e.g., Doctors.

Each tree node contains a binary relationship, which selects the branch to follow to the next node, with the process continuing until a leaf node is reached. The `rpart` function decides whether a leaf node should be split into a condition node and two leaf nodes using a method known as *cost complexity pruning*; any node split that does not improve the overall fit by a factor of CP is not attempted.

A study by Shihab, Ihara, Kamei, Ibrahim, Ohira, Adams, Hassan and Matsumoto<sup>1430</sup> looked at reopened faults in the Eclipse project. Of the 18,312 bug reports 3,903 were resolved (i.e., closed at least once) and 1,530 of these could be linked to code changes. Of the 1,530 that could be linked to code changes, 246 had been reopened at the time of the study. Shihab et al cast their net very wide, extracting 22 factors that could possibly be associated with reopened faults.

Figure 12.2 shows the first few levels of the decision tree, which is visibly very cluttered. The names of the people reporting and fixing problems is part of the fitted model, resulting in some overly long lines (names have been truncated to two characters, so something is visible). The `rpart` package provides basic plotting functionality, and `rpart.plot` package provides much more functionality; the following is the essential code:

```
library("rpart")
library("rpart.plot")

dt=rpart(remod ~ time+week_day+month_day+month+time_days+description_size+
           severity+priority+pri_chng+ num_fix_files+num_cc+prev_state+
           fixer_exp+fixer_name+reporter_exp+reporter_name,
           data=raw_data, weight=data_weight,
           method="class", x=TRUE, model=TRUE, parms=list(split="information"))
rpart.plot(weighted_model, cex=1.2, split.font=1, under.col=point_col,
           box.palette=c("green", "red"), branch.col="grey",
           under=TRUE, type=4, extra=100, branch=0.3, faclen=2)
```

---

<sup>i</sup> Recursive partitioning.

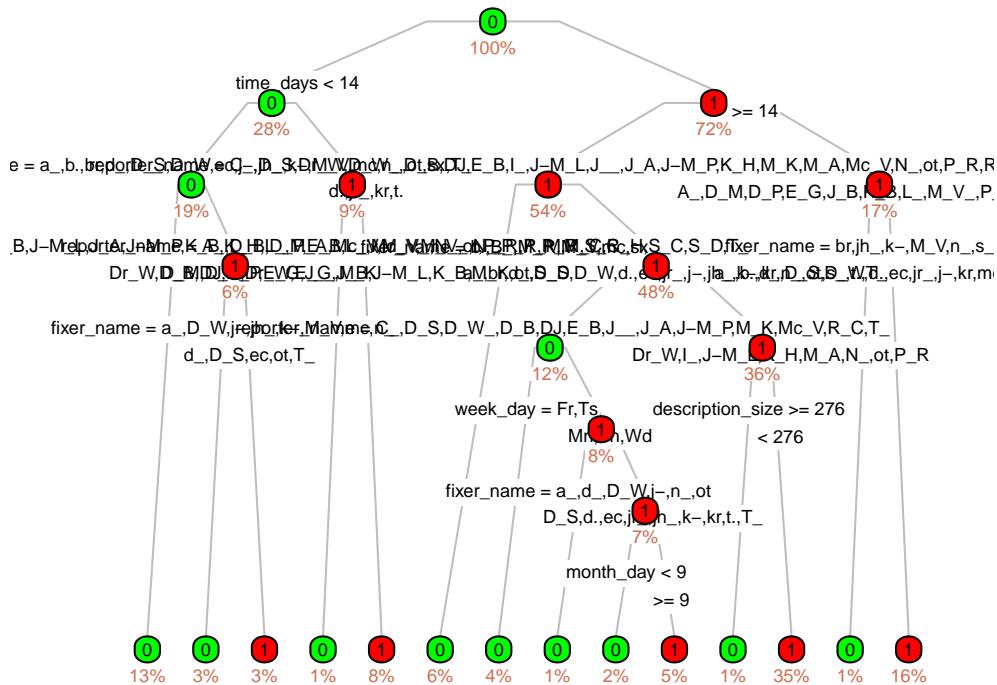


Figure 12.2: Top levels of the decision tree fitted to the reopened fault data (overly long lines are names of people who reported and fixed the fault). Data from Shihab et al.<sup>1430</sup> code

Which variables contribute most to the model? The `summary` (the `cp=0.4` argument removes lots of details from the output; the `printcp` function does not provide any information on variable importance):

```
> summary(weighted_model, cp=0.4)
Call:
rpart(formula = remod ~ time + week_day + month_day + month +
       time_days + severity + priority + pri_chng + num_fix_files +
       num_cc + prev_state + description_size + fixer_exp + fixer_name +
       reporter_exp + reporter_name, data = raw_data, weights = data_weight,
       method = "class", model = TRUE, x = TRUE, parms = list(split = "information"))
n= 1530
```

CP	nsplit	rel error	xerror	xstd
1	0.17010632	0	1.0000000	1.0988118
2	0.02814259	1	0.8298937	0.9773296
3	0.02751720	2	0.8017511	0.8974359
4	0.02095059	5	0.6957473	0.8853971
5	0.01485303	6	0.6747967	0.8975922
6	0.01414947	7	0.6599437	0.9040025
7	0.01407129	9	0.6316448	0.9040025
8	0.01219512	10	0.6175735	0.8941526
9	0.01000000	13	0.5795810	0.8491245

#### Variable importance

	fixer_name	reporter_name	time_days	week_day
32		32	13	6
description_size		fixer_exp	reporter_exp	month_day
4		3	2	2
priority		severity	prev_state	num_fix_files
1		1	1	1
month				
1				

```
Node number 1: 1530 observations
predicted class=0 expected loss=0.4990637 P(node) =1
  class counts: 1284 1279.2
probabilities: 0.501 0.499
```

The variables making the latest contribution to the model and a measure of their relative importance appears at the end (at least when a large `cp` argument is passed).

For the identity of people fixing and reporting problems to play such a large role in the model suggests that either this subset of people have some characteristic or the faults they close

have some characteristic that causes the faults they close to be reopened. A useful pointer for further investigation.

The columns of numbers in the middle of the output contain two measures of error for various values of CP. Decision trees are susceptible to overfitting and the `xerror` column estimates the error using ten-fold cross validation (the error listed in the `rel_error` column does not use cross validation and gives a rosier estimate). The output above suggests that building a model using the CP value listed in the second row is likely to produce more accurate results than other values (the default value of CP is 0.01).

See [odds-and-ends/wcre2012-delaystudy.R](#) for an example of a decision tree analysis of data containing many variables.

## 12.3 Clustering

Clustering is the process of grouping together items (in to a cluster), such that items in a given cluster are more similar to each other than items in other clusters. Some commonly used measure of similarity include distance between items, density of items and distance from a set of items chosen to be representative of some collection of characteristics of interest.

A clustering approach to data analysis requires a method for measuring item similarity and an algorithm for using this information to group the appropriate items within the same cluster. Examples of attempts to understand data, via clustering, include: location based distance Figure 2.18, similarity between Linux distributions based on packages they contain Figure 4.15, trading relationships between companies Figure 4.37, developers contributing to the same Apache project Figure 4.30, density of variables experiencing a given number of read/writes Figure 7.27 and correlation between attributes of Github pull requests Figure 8.9.

A study by Kanda, Ishio and Inoue<sup>820</sup> reverse engineered the evolutionary history of nine large software systems by comparing the similarity of the files containing the source code used to build successive releases. Figure 12.3 shows an unrooted tree representation of the phylogenetic tree estimated from the paired similarity of corresponding files contained in various releases of OpenBSD, FreeBSD and NetBSD.

Archetypal analysis describes individual data points based on the distance from extreme points, whereas cluster analysis focuses on describing its segments using the average members as the prototypes. The `archetypes` package... emailed for data<sup>821</sup>

### 12.3.1 Sequence mining

Recurring subsequences may occur in a collection of related item (or event) sequences. A common application of sequence mining is recommender systems, which are based on finding items that shoppers often buy together, or in sequence (e.g., as children grow up) or shared buying patterns between groups of shoppers.

Common sequences often occur in calls to a given API, e.g., `open/read/close`.

The `arules` package supports the mining of association rules and frequent item sets.

Given  $N$  sequences of items, the fraction of these sequences containing a particular item, say  $X$ , is known as the *Support* for  $X$ . Given that  $Y$  appears in a sequence, what is the likelihood that  $X$  will also appear in that sequence (written  $\{X \Rightarrow Y\}$ )? The following are two common answers:

$$\text{Confidence}\{X \Rightarrow Y\} = \frac{\text{Support}\{X, Y\}}{\text{Support}\{X\}}$$

$$\text{Lift}\{X \Rightarrow Y\} = \frac{\text{Support}\{X, Y\}}{\text{Support}\{X\} \times \text{Support}\{Y\}} = \frac{\text{Confidence}\{X \Rightarrow Y\}}{\text{Support}\{Y\}}$$

A study by Fowkes and Sutton<sup>521</sup> investigated the sequence of API calls, made within the method bodies of 17 Java systems. The following call to `apriori` searches for sequences of method calls (in the `drools` business management system) having a given *support* and *confidence*; see [odds-and-ends/drools.R](#):

```
library("arules")

drools=read.transactions(paste0(ESUR_dir, "odds-and-ends/drools.csv"),
                         format="single", cols=c(1, 2))

rules=apriori(drools, parameter=list(support=0.0001, confidence=0.1))

summary(rules)
inspect(head(rules, n=3, by = "confidence"))
```

The number of rules of a given length found were (lhs and rhs refer to the two sides of the  $\Rightarrow$  symbol):

rule length	distribution (lhs + rhs):sizes
2	1478
3	252
4	32
5	5

The apriori algorithm treats each item, in a sequence, as being independent. The order of method calls may be significant and ??? can be used to introduce constraints on item sets.

A particular sequence of method calls may be missing a necessary call; the recommenderlab package provides support for developing and testing recommendation algorithms.

## 12.4 Ordering of items

Arranging items in order can reveal information about the form of the calculation used to select the relative positions of ordered items. Given multiple orderings of the same items, ordering patterns of subsequences of items, common to multiple sequences may indicate a shared semantics for those items.

### 12.4.1 Seriation

Seriation is the process of placing items into a linear order, based on a metric derived from the characteristics of the items. The number of possible item orderings grows as  $n!$ , making it impractical to evaluate every possibility for non-trivial sample sizes; various heuristic algorithms have to be used. The seriation package supports a variety of functions that attempt to find an optimal linear ordering of items.

A study by Jones<sup>789</sup> investigated the extent to which developers create similar data structures to hold information listed in a specification, e.g., grouping together identifiers containing related information in the same data structure. The hypothesis was that shared cultural and professional experiences would result in subjects defining data structures containing similar contents.

Subjects were given a list of items from the “Department of Agriculture” and asked to design a C/C++ API containing this information. The results, from each subject, were the structs or classes defined and their fields/members (with each field containing one item of API information, e.g., “Date crop harvested” and “Organically produced”).

Ordering of the data highlights the information that tends to be colocated in the same data structure, and the subjects making similar choices.

Figure 12.4 shows which items were placed in the same data structure as the information item “Antibiotics used”, by each subject (colored squares indicate presence). The matrix passed to seriate contains boolean values (indicating presence in the same data structure) and the subjects/fields are ordered so that shared usage appears adjacent. The bertinplot function provides a particular visualization of the ordered data.

```
library("seriation")

fser=seriate(fmat, method="BEA", control = list(rep = 10))
bertinplot(fmat, fser, options=list(panel=panel.squares, spacing=0,
                                    gp_labels=gpar(cex=0.6)))
```

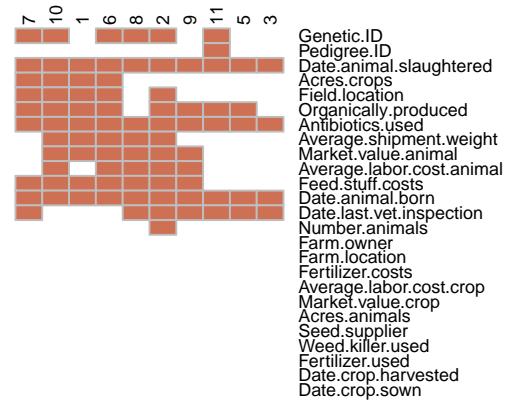


Figure 12.4: A Bertin plot for items included in the same data structure as the item “Antibiotics used”, for each numbered subject, after reordering by seriate. Data from Jones.<sup>789</sup> code

A single item's pattern of association, with all the other items, can be generalised by counting occurrences of every pair of items in the same data structure. A Robinson matrix has the property that the value of its matrix elements decrease, or stay the same, when moving away from the major diagonal; this matrix has been used to study commonality in subjects' categorization behavior.<sup>1350</sup> Figure 12.5 shows a visualization of a Robinson matrix for the Jones data.

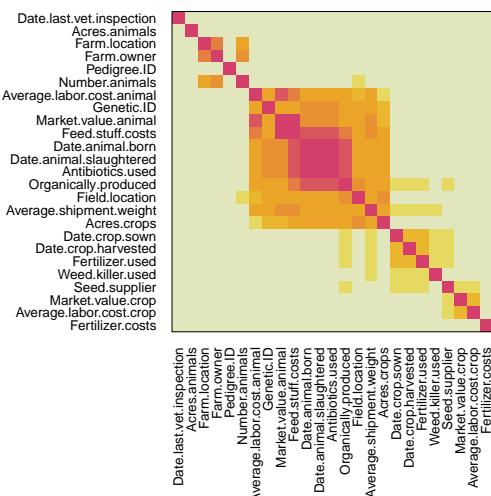


Figure 12.5: A visualization of the Robinson matrix based on number of times pairs of items co-occur in the same data structure (the closer to the diagonal the more often they occur together). Data from Jones.<sup>789</sup> [code](#)

```
library("seriation")

fdist = as.dist(1 - fmat/max(fmat)) # Normalise counts
fser = seriate(fdist, method="BBURCG")

pimage(fdist, fser, col=pal_col, key=FALSE, gp=gpar(cex=0.8))
```

## 12.4.2 Preferred ordering

A list of items may have a preferred ordering. The ordering may be the result of rankings, rating or a comparison between pairs of items.

Bradley-Terry statistics are the traditional technique for calculating an order for a list of items, based on results from pairwise comparisons, e.g., football match results (there is an extension for analyzing results from simultaneous comparisons of more than two items). This section is based around use of the BradleyTerry2 package for simple paired comparisons and the prefmod<sup>663</sup> package for the more complicated cases.

An alternative approach to analyzing item ordering is discussed in Section 9.6.1.

A study by Jones<sup>786</sup> investigated developer beliefs about binary operator precedence. Subjects saw an expression, such as  $a + b \% c$ , and were asked to insert parenthesis such that the behavior (as interpreted by the compiler) remained unchanged. What is the relative precedence of the binary operators used in the study, based on subject answers (which may be different from the actual precedence)?

The parenthesis specifies the operator that can be treated as *winning* the precedence choice. The BTm function, in the BradleyTerry2 package<sup>1571</sup> takes the results from paired comparisons, and returns the coefficients of a fitted model (internally, the glm.fit function is used to fit a logit regression model).

```
library("BradleyTerry2")

prec_BT=BTm(cbind(first_wl, second_wl), first_op, second_op, data=nodraws)

summary(prec_BT)

plot(qvcalc(BTabilities(prec_BT)), col=point_col, main="",
      xlab="Operator", ylab="Relative order")
```

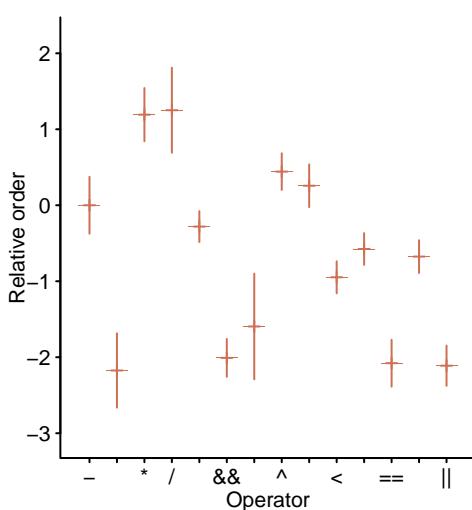


Figure 12.6: Relative ordering of binary operator precedence, with corresponding error, based on subject responses to precedence questions. Data from Jones.<sup>786</sup> [code](#)

The output produced by summary has the same form as that produced by other regression models. Note: the summary function does not list the factor with value zero (the subtraction operator, for this data). The BTabilities function lists all factors and their values, and the call to plot uses this information to a visualization of the estimated coefficients and the corresponding error.

Figure 12.6 is a visualization of the estimated coefficients, which provide an estimate of relative binary operator precedence, along with a corresponding error estimate; the difference between coefficients can be interpreted as an estimate of the strength of subjects' belief in the difference.

In a sports match the home team is often considered to have an advantage over the away team. Perhaps, when developers are uncertain they are more likely to select the first binary operator, of a pair. A model can be updated to include information on first position wins, for each operator pair (the effect is very small); see [developers/jones\\_prec.R](#) for details.

Three or more items can be compared by ranking them in order of preference. The pattr.fit function, in the prefmod package, can be used to analyze ranked data (and the pattr.worth function to map it to a plotable form).

A study by Biegel, Beck, Hornig and Diehl<sup>160</sup> investigated the ordering of definitions appearing in 5,372 classes, from 16 Java programs. The Java coding conventions (JCC)<sup>1520</sup> recommends a particular ordering of the four kinds of definitions, and on average over 80% of classes in these projects follow the JCC recommendation: *class variable* (or *field*), *instance variable* (or *static initializer*), *constructor* and *method*.

Definitions include access control information, via the use of the keywords: *private*, *protected*, *public* and no keyword (the default behavior given in the language specification is used). The ordering of definitions in the source code, by identifier visibility (i.e., access control), can be treated as a ranking.

Figure 12.7 shows the ordering (ranking) of definitions by access control keyword, over all projects investigated; see [sourcecode/member-order/vis-key\\_order-pref.R](#) for details (most of the code is needed to convert the data to an array of rankings, needed by the `pattR.fit` function).

The y-axis values, labelled "Worth estimate", can be used to estimate the probability that an identifier defined with visibility  $V_i$  appears before one with visibility  $V_j$  using the formula:  $\frac{V_i}{V_i + V_j}$ .

An item may be given a rating or score, say out of 10, rather than compared to one or more other items. These ratings can be used to assign a relative ordering to the rated items.

A study by Dorn<sup>7</sup> asked subjects to rate code snippets for readability, using a five point scale. The 5,468 subjects were asked to rate 20, out of 360 possible, code snippets. bug in prefmod...?

## 12.5 Simulation

Simulating a process or system, via an executable model, is a means of gaining information about the operational characteristics of the process or system (assuming the characteristics of the model are sufficiently accurate). Varying the characteristics of a model can provide possible answers to what-if questions. Some of the kinds of simulation methods available include:

- discrete event simulation: the system modeling is based on the discrete events that can occur... The `simmer` package...
  - Staff scheduling... [projects/impl-sim.R](#)
- agent-based modeling: a set of agents, having specified attributes, interact with each other in a computer simulation. After a given number of time steps the state of the system is measured, with the results from many simulation runs combined to calculate probabilities for the various end-states. NetLogo is a widely used system for agent-based modeling and the `RNetLogo` package provides an R interface; see [odds-and-ends/peter-principle.R](#).
- system dynamics: here a system is represented by causal loops between all the interacting components, essentially an analogy representation of differential equations. System dynamics has been used to simulate software project staffing<sup>227</sup>...
- differential equations: if a system can be described using a set of differential equations, the `deSolve` package can be used to numerically solve these equations.

Fitting sales data to the Bass diffusion model was discussed in Section 3.6.3. Duggan<sup>416</sup> investigated the impact, on sales volume, of the spatial separation of potential customers living in 10 regions (Figure 12.8 shows the connections between regions and the percentage of total population they contain, given as initial conditions in the numerical solution). See [odds-and-ends/RJ-2017.R](#) for implementation details.

- BUGS, JAGS, Stan...

?

social influence...?

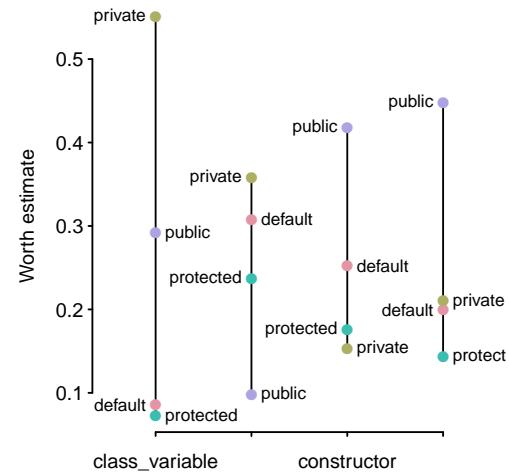


Figure 12.7: "Worth estimate" of identifier access control (visibility), for the four kinds of definitions that can appear within a Java class. Data from Biegel et al.<sup>160</sup> [code](#)

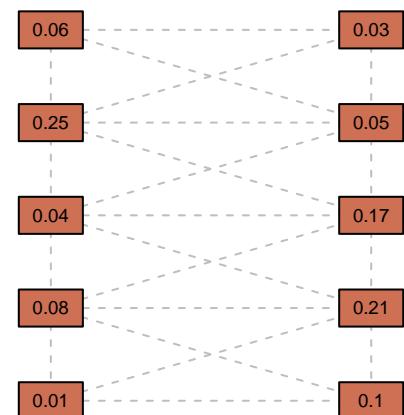


Figure 12.8: Region populations and their connections: initial conditions used in Duggan's<sup>416</sup> numerical solution of the Bass equation. [code](#)



# Chapter 13

## Experiments

### 13.1 Introduction

Does doing X have a significant effect on S? Traditionally X might have been a new kind of fertiliser (or drug) and S the crop yield (or being cured of some illness). In software engineering the effect sought is often a performance improvement and X the latest snake oil.

In an observational study the researcher is a passive observer, simply recording what happened, or is happening. In an experimental study the researcher actively attempts to control the values of the explanatory variables (a common technique is to vary the values of one explanatory variable, while the others are held constant; in some cases, the environment in which events occur form a natural experiment, in that the explanatory variables of interest vary in a way that an experimenter might vary them).

A controlled experiment is the technique used to obtain the data needed to test a hypothesis. The controlled experiment that most developers are likely to be familiar with is benchmarking.

Dramatic changes in performance, after doing X, are relatively common in software development and in such cases using statistics to confirm that a noticeable change has occurred is almost a formality. At the other extreme, if a difference requires statistical analysis to be detected, it might not be a difference worth being concerned about in practice.

Advice for running experiments, often follows the waterfall model of software development, with lots of upfront planning and little or no feedback from production use until the end of the process. This advice has its roots in the environment in which experiments are carried out by the audience of many statistics text books, where running an experiment is costly (in money or time), or is a once only opportunity.<sup>i</sup>

Some experimental questions in software engineering are amenable to iteration. Running quick, inexpensive experiments, can be a cost effective technique for filtering possible issues of interest, and obtaining information on which, of the myriad of variables, have a large impact on the response variable.

Finding the right question to ask is sometimes the most useful output from running an experiment.

An important point to remember is, that, it is better to have an inexact answer to the right question than an exact answer to the wrong question.

Like all software development activities, experiments have to pay their way. Some of the answers needed for a cost-benefit analysis include: the cost of running an experiment, capable of producing the information of interest, within acceptable confidence intervals; the usefulness of the data likely to be obtained, by running an experiment, using a given amount of resources (such as time and money).

Many software engineering tasks are performed within complex environments. Controlling and measuring all the variables in the environment is so time-consuming and expensive

---

<sup>i</sup> Experiments in the social sciences, major producer of experimental studies, are often grant funded, with limited opportunities for rerunning experiments that failed to produce data that can be published.

that few researchers have been willing to attempt realistic controlled experiments. Consequently, much of the hypothesis testing performed in commercial environments has been based on convenience samples, obtained from experimentally uncontrolled, production software projects.

Running an experiment with minimal funding means that experimental subjects are often unpaid volunteers, from a pool containing who ever is available.

Software engineering is not known as a research area where experiments are commonly performed. A study<sup>1451</sup> of 5,453 papers in software engineering journals, published between 1993 and 2002, found that only 1.9% reported controlled experiments (of which 72.6% used students, only, as subjects) and the statistical power of many of these experiments fell below expected norms.<sup>424</sup>

### 13.1.1 Measurement uncertainty

The term *measurement error* is often applied to measurements involving physical quantities. It is based on the assumption that any difference between the measured and actual value is caused by errors made in the measurement process.

In software engineering, some quantities can be measured exactly, e.g., lines of code in a source code file. However, the process that generated the code (e.g., a software developer) may produce a different number of lines, if it is repeated using a different developer or even the original developer (reimplementing the program); see Figure 5.20. The code that is measured is a sample from a population, and measurements involving this code must be treated as having an accompanying uncertainty.

Goodhart's law (which is an observation about human behavior, rather than a law) is "Any observed statistical regularity will tend to collapse once pressure is placed on it for control purposes." If the measurements collected were actively used to control or evaluate the development team, then the developers would be motivated to move the measurements in a direction favorable to them.

A study by Perry, Staudenmayer and Votta<sup>1236</sup> investigated differences in the measurements of various software development activities (e.g., working time on an activity and activities performed throughout the day), as measured by the developers involved (i.e., self reports) and as measured by external observers. Differences included, developers reporting activity time that was 2.8% greater, on average, than reported by an observer; the agreement rate on activities performed varied between 0.6 and 0.95.

A series of studies<sup>841</sup> of social network data, as reported by those within the network and extracted from externally observed information, found that differences were large enough to render invalid any analysis of a network characteristics based on member supplied information.

Random variability in the performance of, what are intended to be, identical hardware components, is discussed in Section 13.3.2.1.

## 13.2 Design of experiments

Randomization is the foundation on which any claims of causation, involving experimental results, are based (i.e., doing X caused Y). Without randomization, the most that can be said is that a correlation between doing X, and Y occurring, has been observed.

The purpose of running an experiment is to obtain data that can be used to help answer one of more questions. Experiments have to be carefully designed, to ensure that the data obtained is representative of the processes involved in the question(s) being asked. Design issues include:

- recruiting the appropriate subjects from the available population (sampling is discussed in Section 10.2),
- creating an experimental task(s) that shares all the important characteristics of the tasks associated with the questions of interest,

- creating an environment for the subjects, in which they can give a suitable performance, during the experiment.
- Subject characteristics can sometimes interfere with good experimental design, e.g., human subjects have a memory of their previous experiences that they cannot choose to erase,
- controlling all variables that could have a significant impact on the response variable(s) of interest. Failure to take into account, and control, variables having a significant impact, can cause a tiny effect to appear to be a large effect and vice versa. One person's tongue-in-cheek-advice on how to bias an experiment to get the desired outcome<sup>1087</sup> is another person's list of thoughtless mistakes.

When factors cannot be controlled, they need to have their impact contained. One technique for handling the problem of uncontrolled variables is to group subjects into blocks based on the variable that is suspected of influencing the response (a process known as *blocking*), randomization of subjects then occurs within each block. The identity of the block becomes another explanatory variable during analysis of the results.

A study by Basili, Green, Laitenberger, Lanubile, Shull, Sørumgård and Zelkowitz<sup>109</sup> compared the performance of perspective based reading (which instructs reviewers to read a document from a specified perspective, e.g., a designer, tester or user) against the reading technique currently used by the professional developers who were the subjects.

The researchers thought it likely that, training subjects to use the new technique would change subject performance on whatever technique they currently used, and decided to measure subject performance using their existing review technique first, before giving subjects training in the new, perspective based reading, technique.

Have all subjects use the same techniques, in the same order, means it is not possible to separate out ordering effects in the results, e.g., effects such learning during the experiment and any random distraction effects that only occurred at certain times.

Some of the factors outside of the control of the researchers, which could affect the results, include:

- the time taken for people to become proficient at using a new technique, old habits die hard. How much practice do subjects need, for them to be able to give a performance that makes it possible to reliable compare a new technique? In this study subjects were taught PBR two days after the first part of the experiment, they trained on a test document, reviewed one document, received more training and then reviewed another document.
- the kind of review technique used by subjects in the first half of the experiment. A change in performance is expected, but it is not known what technique the change is relative to (it is assumed that adhoc techniques are being used),
- the characteristics of the seeded faults. Were more faults found in the NASA documents because readers were familiar with reading that kind of document, or perhaps the characteristics of the seeded faults was such that they were harder to detect in one kind of document than another?

The experimental output included, for each subject, the number of faults detected (which has a known upper limit and a yes/no detection status) and the number of false positives in each document reviewed (which has no upper limit, in theory). See [faults/basili/pbr-experiment.R](#) for details of fitting a regression model.

Basing all experimental choices on random selection does not automatically create samples that maximise the information that can be obtained.

A study by Porter, Siy, Mockus and Votta<sup>1264</sup> investigated software inspections. The structure of the inspection process was manipulated by varying the number of reviewers (1, 2 or 4), number of meeting (1 or 2) and for multiple meetings whether reported faults were repaired between meetings (88 inspections occurred, involving 130 meetings and 17 reviewers).

Selecting the treatment to use, for a review, from successive entries on a randomised list of all possible treatment structure combinations (created at the start of the study), would ensure that the results are balanced across the variables of interest. However, the choice of treatment to use was randomly selected from all possibilities, as each unit of code became

available for review, resulting in some combinations of reviewers/meetings/repaired not being used and some used very often. The results contain an unbalanced set of experimental conditions, making it difficult to reliably fit a model; see [experiment/porter-siy/inspection.R](#), [experiment/porter-siy/meeting.R](#) and Figure 11.32.

### 13.2.1 Subjects

Experimental subjects might be people or artefacts (e.g., computers). An essential requirement for generalising the results from an experiment, to a larger population of subjects, is that the characteristics of the experimental subjects are representative of the applicable characteristics of the population of interest.

The major issues involved in having computer hardware as an experimental subject are covered in Section 13.3, while the major issues in human cognitive performance are covered in Chapter 2. A limiting factor, when designing an experiment involving human subjects, is typically the amount of time the subjects are likely to be willing to make available to participate.<sup>1450</sup>

When people are the subjects in experiments, a variety of human factors introduce uncertainty into the results; people constantly adapt to their environment, including the environment of an experiment (e.g., they learn and retain memories of their experiences; see Section 2.5), they also experience fatigue, and their attention ebbs and flows during an experiment.

Professional developers, working within different ecosystems, may share a set of basic skills and knowledge, such as being able to fluently use at least one programming language.

Much of the published research involving human subjects, in software engineering experiments, has used students. Students are a convenience sample for many researchers, with results based on student subjects being accepted for publication by some journals. Industry is well aware that students' software engineering skills are not representative of professional developers (who have a few years experience); industry is where many graduates find employment after graduation, and the abilities of these new employees is plain for everyone in industry to see.

- students' commercial software skills and knowledge is likely to be very poor, in comparison to professional developers.<sup>1116</sup> This lack of experience and know-how means that student subjects need to spend time on activities that are second nature to professionals, or they simply make noncommercial judgement calls,
- students, typically, have very little experience of writing software, perhaps 50 to 150 hours (and many have no basic coding skills<sup>972, 1039, 1581</sup>), while commercial software developers are likely to have between 1,000 to 10,000 hours of experience. This lack of programming fluency means that student programming performance is likely to contains a large learning component, as well as student performance being much lower than professional developers,<sup>1102</sup>

In other research areas students subjects may be more representative of the target population, because they have had many years of experience performing the activities and tasks used in those areas, e.g., processing text written in English and everyday image processing, which are activities used in cognitive psychology experiments.

When experimental results are intended to be applied to the population of university students studying a software related subject, subjects drawn from this population can be representative.

In the US, UK and some other countries, students pay to attend university, and like all businesses universities have to respond to customer demand. When a student decides to study a computing related subject, the University's interests are in ensuring that the student meets its minimum entry requirements and can pay; the likelihood of that person being offered employment in a software related job is not a consideration (in the UK computer science graduates have a much higher unemployment rate, six-months after graduation, compared to those studying other STEM subjects<sup>1409</sup>). Given the high failure rate for computing degrees<sup>1678</sup> and introductory programming courses,<sup>1642</sup> many students on such courses may not even have any software development skill or ability...

Note on terminology: many academic studies use the phrase *expert* to describe subjects who are final-year undergraduates or graduate students, with the term *novice* used to describe first-year undergraduates. In a commercial software development environment a recent graduate is

considered to be a *novice* developer, while somebody with five or more years of commercial development experience might know enough to be called an *expert*.

Amazon's Mechanical Turk is becoming popular as a resource for finding subjects and running experiments (in 2015 the population of workers was estimated to be 7,300<sup>1508</sup>). Subjects can stop taking part in a MTurk experiment at any time and care needs to be taken to ensure that the characteristics of subjects who remain does not bias the results.<sup>1710</sup>

### 13.2.2 The task

Generalizing experimental results to daily work conditions, requires that the characteristics of the tasks performed by subjects share the essential characteristics of the work tasks. That is, the task needs to mimic realistic activities (the technical term is being *ecologically valid*):

- being representative of real world intended usage requires information about how a system will be used in practice, along with the inputs it is likely to experience; lack of resources to perform an analysis of real world usage often means that a convenience sample is used. In a rapidly changing environment, it may not be possible to specify usage patterns in sufficient detail, and perhaps one of the important real world behaviors that needs to be benchmarked is adaptability to change.

A study by Gregg and Hazelwood<sup>624</sup> provides an example where data usage characteristics are the deciding factor in a cost/benefit trade-off. The time taken to perform a matrix multiply, when the CPU uses a local GPU (the SGEMM implementation in the nVidia CUBLAS package<sup>1174</sup> was used) was measured. Figure 13.1 shows that moving data between the CPU and GPU consumes a significant amount of time, relative to the work done on the data once inside the GPU. Estimating whether GPU usage is worthwhile, depends on the size of matrices encountered in the real world use case, the performance may be slower because of the data transfer overhead, or faster if the matrices are large enough to consume the larger amount of compute resources.

Another example of the impact of variations in the input data relates to Figure 10.9,

- Products that appear to be very similar can have very different performance characteristics (this is one reason why they exist as different products; the other common reason is marketing). Using a product that is identical to the one used in production avoids this problem.

In a study by Bird,<sup>165</sup> a performance optimization expert took the existing generic code of a library and created tuned versions for each of five different processors (IBM's Blue Gene P and four different members of Intel's x86 product line). The performance of the generic, and all tuned versions of the code, was measured on all processors. Figure 13.2 shows relative performance, with the x-axis listing the processor the code was tuned for and the y-axis the processor on which it was run; the numbers are relative performance difference, compared to running code on the processor for which it was specifically written.

Availability of resources is often a constraining factor, for running an experiment that mimics real world usage. For example, benchmarking backup/restore tools, or desktop search applications, requires realistic file system contents (e.g., the file system must contain a realistic number of files, directory depth, disk fragmentation, etc); getting to a position of being able to generate realistic file systems is a non-trivial task,<sup>13</sup> let alone realistic file content characteristics.<sup>1537</sup>

### 13.2.3 What is actually being measured?

Subjects may not solve the problems they are presented with, in an experimental, in ways that were intended by the person who designed the experiment.

The history of research into human memory provides an example of how early experimental results were misinterpreted.<sup>784</sup> These early experiments asked subjects to remember sequence of digits, the results suggested that short term memory has a capacity limit of  $7 \pm 2$  items.<sup>1083</sup> After many years and more experiments, the  $7 \pm 2$  digit limit model was replaced by a model based on a limit of 2 seconds of sound<sup>88</sup> (in English this corresponds to around 7 digits, 5.8 in

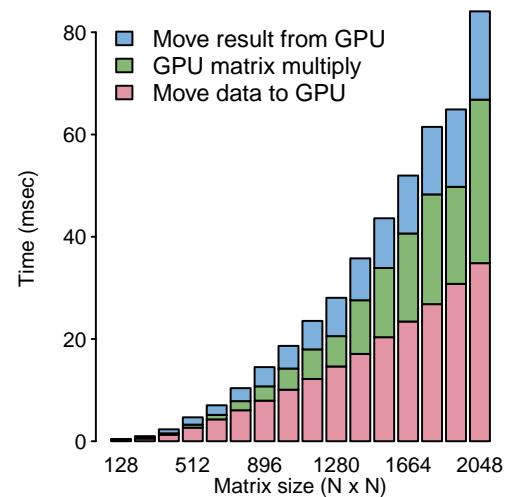


Figure 13.1: Time taken to transfer and multiply 2-dimensional matrices of various sizes on a GTX 480 GPU. Data kindly supplied by Gregg.<sup>624</sup> code

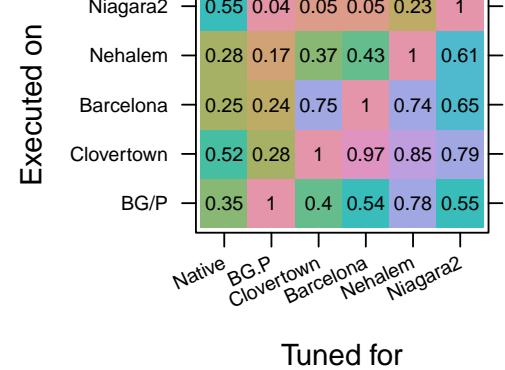


Figure 13.2: Relative performance (y-axis) of libraries optimized to run on various processors (x-axis). Data from Bird.<sup>165</sup> code

Welsh<sup>443</sup> and around 10 digits in Chinese:<sup>720</sup> the number of digits that can be held in memory when people use these languages).

Software developers are problem solvers and get plenty of practice in finding patterns that can be used to achieve a goal. Unless an experiment is carefully constructed, it is naive to assume that developers will use any of the techniques anticipated by the person designing the experiment.

Your author once ran several experiments,<sup>786</sup> expecting to find a *2 seconds of sound* effect in developers short term memory of source code (sequences of simple assignment statements were used). A great deal of effort was invested in creating code sequences whose spoken form required either more or less than 2 seconds of sound, but the results did not contain any evidence of the expected effect (i.e., a difference in performance caused by the length of sound in the spoken form of source code statements).

At the end of one experiment, a subject mentioned a strategy used to help improve his performance: remembering the first letter of each variable (your author had not noticed that the variables in each list had unique first letters). Use of this strategy reduced the amount of STM the subject needed to use, providing one explanation why the expected effect was not found. The last task, on subsequent experiments, asked subjects to list any strategies they had used during the experiment.

What appear to be small differences, can have a large impact. Human written source code contains very similar constructs where, what might be thought to be small differences in semantics, have usage patterns which are very different. For instance, many languages allow numeric literals to be specified using decimal and hexadecimal notation; Figure 13.3 shows that there is a large difference in the distribution of occurrence of literal values written using each notation (at least in C source).

A study<sup>638</sup> of how subjects split identifiers, in source code, into components and expanded them, or not, into words, measured individual performance against what was considered to be the definitive expansion of each identifier. The results of this experiment could also be used to measure the performance of the researchers, in creating a list of identifier expansions that maximised the likelihood of developers correctly decoding the intended information.

Subject motivation is an important factor in obtaining reliable experimental data. Subjects who feel they are being coerced may respond by providing spurious responses, or simply attempt to find short-cuts that minimise the time then need to spend taking part in the experiment, without attracting attention by making too many mistakes.<sup>668</sup> Your author always requests that subjects put as much effort into performing the task, as they would at work: not more, not less, effort.

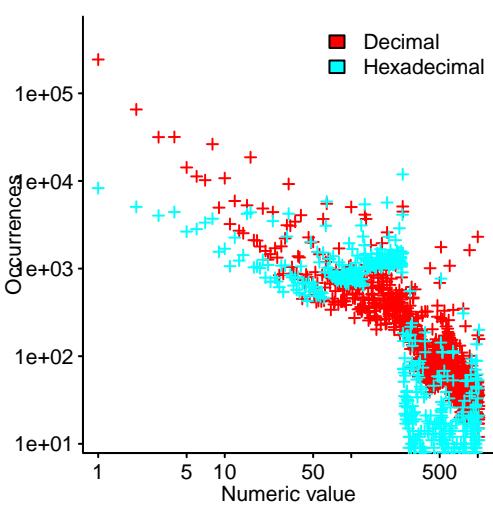


Figure 13.3: Number of integer constants, appearing in the visible form of C source code, having the lexical form of a decimal-constant (the literal 0 is also included in this set) and hexadecimal-constant that have a given value. Data from Jones.<sup>785</sup> code

### 13.2.4 Adapting an ongoing experiment

The cost of running an experiment makes it very tempting to stop, as soon as what is thought to be enough data is obtained, to produce a reliable result. For instance, a researcher may process subjects in batches, running statistical tests after each batch of results, to check that everything is Ok.

Differences in subject performance, as an experiment progresses, will cause some fluctuation in the analysis of the currently available data, making it possible that some cut-off value (e.g., a p-value significance level) is temporarily achieved, before later data causes it to revert to a less extreme value.

Stopping conditions... data...

Examples of ways in which A/B testing on the web can go wrong<sup>873</sup> ...

When running an experiment on a live system (or live people), an analysis of the ongoing measurements may suggest that certain changes are likely to have a worthwhile impact. Adaptive designs is the term used for experimental designs that support modification of an experiment as results become available.

An adaptive design might be used to reduce the number of different combinations that need to be measured, e.g., benchmarking a computing system supporting many options.<sup>1287</sup>

### 13.2.5 Selecting experimental options

The behavior of systems may be configured by selecting from a variety of options, and an estimate of the impact of individual options, on system performance, may be required. One way of obtaining this information, is to measure system performance for all possible combinations of option values. This approach might be practical for a small number of options, each having relatively few values, e.g., Apache supports nine build time yes/no options giving  $2^9$  possible configurations (out of these 512, only 192 are valid). However, large systems often support so many options, that building and executing every configuration would be impractical (e.g., SQLite supports 3,932,160 valid options).

An experiment in which all possible permutations of option values are tested, is known as a *full factor design* (options go by the experimental term *factors*). The `fac.design` function, in the `DoE.base` package, takes a specification of the factor levels and returns a list of all combinations that need to be run to perform a full factor design; see [experiment/design\\_fac.R](#).

A study by Citron and Feitelson<sup>298</sup> investigated the performance impact of adding, what they called a Memo-Table (essentially a cache designed to store and reuse the results of previously executed instruction sequences), to the IBM Power4 cpu architecture. The configuration options for the Memo-Table were Size (1k or 32k), Associativity (1-way or 8-way), Mapping (indexing by program counter or operand+opcode) and Replacement method (random or least recently used).

Four configuration parameters, each having two possible values, gives  $4^2 \rightarrow 16$  possible configurations. Citron and Feitelson benchmarked all 16 possibilities, enabling them to check for interactions between all factors. In many experiments the number of interactions between factors is small, and a common cost saving is to only consider interactions between pairs of factors (rather than, say, between three factors).

Factor having just two possible values is a common case, and is known as a two-factor factorial design: it is a *full two-factor design* when all combinations used and a *fractional two-factor design* when a subset is used.

A study by Lee and Brooks<sup>934</sup> involved three optional values per parameter; see [experiment/lee2006/lee.R](#).

The `FrF2` function, in the `FrF2` package, generates a list of the combinations of factor values that need to be run, to analyse  $N$  factors having a resolution of  $R$  (the ability to separate out main effects and interactions between factors; to be able to separate out main effects a resolution of 3 is required, a resolution of 4 enables detection of separate pairs of interactions). In the following list, 1 indicates the option is enabled, -1 that it is disabled; the output from some functions uses +/-, rather than 1/-1:

```
> library("FrF2")
> FrF2(nfactors=4, resolution=3, alias.info=3)
  A  B  C  D
1  1 -1  1 -1
2 -1 -1 -1 -1
3  1  1 -1 -1
4  1 -1 -1  1
5 -1  1 -1  1
6  1  1  1  1
7 -1 -1  1  1
8 -1  1  1 -1
class=design, type= FrF2
```

The price paid for running a fractional, rather than full, factorial design experiment, is that it is not possible to distinguish interactions between some combinations of factors. For instance, after running the eight combinations listed above, it is not possible to distinguish between an effect caused by a combination of the AB factors, and one caused by the combination CD; this combination is said to be *aliased*. The complete list of aliased factors is:

```
> design.info(FrF2(nfactors=4, resolution=3, alias.info=3))$aliased
$legend
[1] "A=A" "B=B" "C=C" "D=D"

$main
[1] "A=BCD" "B=ACD" "C=ABD" "D=ABC"
```

```
$fi2
[1] "AB=CD" "AC=BD" "AD=BC"

$fi3
character(0)
```

To distinguish between an effect caused by any of these combinations, all 16 factor combinations have to be run.

Factorial designs require the number of runs to be a power of two, so the number of different runs grows very quickly as the number of factors increases.

A Plackett and Burman design requires that the number of runs be a multiple of four and at least one greater than the number of factors (they are non-regular fractional factorial 2-level designs). The down-side of these designs is that the results from experiments using them will only support the analysis of the main factors, i.e., any interactions between factors will not be detected; also Plackett and Burman designs can contain complex aliasing between the main factors and (possible) interactions between pairs of factors. The `pb` function, in the `FrF2` package, generates Plackett and Burman designs.

Multiple fractional factorial designs can be combined to isolate effects (i.e., remove aliasing between combinations of factors). Some signs in the original design are switched, creating what is known as a *fold over* of the original; switching the signs of all factors is known as a *full fold over*. The `fold.design` function generates a foldover design from an existing design.

Randomly selecting option values is an inefficient use of resources, because some option values are over/under used (see [experiment/SQLPWR.R](#) from a study by Guo et al<sup>643</sup>).

### 13.2.6 Factorial designs

A variety of techniques are available to help visualise the results from an experiment using a factorial design. The analysis is the same for full and partial fractional designs, the only difference is the number of interactions between factors that will be available for analysis.

The study by Citron and Feitelson,<sup>298</sup> discussed earlier, used the SPEC CPU 2000 benchmark, and the values measured were integer floating-point performance, power consumption, processor timing and die area of the chip.

For simplicity consider three of the factors (ignoring, for the time being, the replacement method), the results can be visualised as a cube with each of the eight vertices representing one combination of factor values; the values at each vertex of Figure 13.4 are the SPEC benchmark cint performance figures.

Imagine taking two opposite faces of the above cube, say the two for size on the left and right going into the page, and finding the mean cint value for both faces, the difference between these two values is known as the *main effect* for size; the main effect for the other factors is similarly calculated.

A design plot is a visualisation of these main effects, with a central horizontal line showing the overall mean value of the response variable. Figure 13.5, created using the `plot.design` function, shows the impact that each factor can have on the value of cint, offset from the mean value.<sup>ii</sup>

For an example involving more changeable parameters; see [experiment/lee2006/lee.R](#).

At a finer level of granularity, an interaction plot shows the interaction between pairs of factors. Figure 13.6 shows how the mean value of cint varies as size is changed, for a given value of mapping (see legend).

For an equation based analysis, a fitted regression model can be used to investigate the interactions between factors. For instance, the following model specifies interactions between all variable pairs; see [experiment/MemoPower03.R](#) for details:

```
Memo_glm=glm(cint ~ (size+associativity+mapping)^2, data=Memo)
```

<sup>ii</sup> `plot.design` makes some unexpected display decisions when the explanatory variables are not factors.

A study by Pallister, Hollis and Bennett<sup>1208</sup> investigated the power consumed by various embedded programs when compiled with gcc using various command line parameters. The design used contained partial aliases, which many plotting functions cannot handle; the half normal function, from the DoE.base package, has an option to orthogonalize the design; see [experiment/pallister/gcc-power.R](#).

Plackett and Burman designs do not contain enough information to fit a regression model, a bespoke method has to be used, e.g., the DanielPlot function, in the FrF2 package (in the plot produced, the x-axis shows effect size, the y-axis contains diagnostic information). If the data is the result of random variation (i.e., changing factor values has no effect), differences between pairs of factor averages have a (roughly) normal distribution; plotting values from a normal distribution using a normal probability scale produces a straight line. If many of the points displayed by DanielPlot appear to form a straight line, then the corresponding factors are likely to have had little effect on the results; any factors well off the line are of interest.

A study by Debnath, Mokbel and Lilja<sup>376</sup> investigated the impact of seven system configuration settings on PostgreSQL performance, on the TPC-H benchmark. High and low values were chosen for the configuration values and a Plackett and Burman design with full fold-over was used. Figure 13.7 shows the half-normal plot from the 16 runs; factors P4 and P7 do not fall on a straight line that passes close by the other factors, and they also exhibit the largest effect.

## 13.3 Benchmarking

Benchmarking is the process of running an experiment to obtain information about the performance of some aspect of hardware and/or software. Common reasons for benchmarking, in software engineering, include comparing before/after performance and obtaining numbers to put in a report. For a more general audience, benchmarking information is often used as input to a selection process and as such often has a marketing orientation. Accurate measurements are not always necessary, showing that a system is good enough, may be good enough.

The time taken for operations such as add and multiply was used to compare the performance of early computers.<sup>169, 1056, 1649, 1650</sup> Studies by Knight<sup>868, 869</sup> calculated performance based on a weighted average of instruction times, based on the kinds of instructions executed by commercial and scientific programs. Based on a study of over 300 computers available between 1944 and 1967, the rental cost for performing an operation decreased with increasing computer performance; see Figure 13.8, the lines are fitted power laws with exponents between two and three; also, see [benchmark/EvolvingCompPerf\\_1963-1967.R](#).

Obtaining accurate benchmark data for many questions relating to computing platforms is may to be economically infeasible.<sup>iii</sup> A consequence of the continual reduction in the size of components within microprocessors (see Figure 13.9 and Figure 11.50), is that individual components are now so small that variations in the fabrication process (e.g., differences in the number of atoms added or removed during the fabrication process) can noticeably change their geometry, leading to large variations in the runtime electrical characteristics of supposedly identical devices.<sup>152</sup>

Manufacturers offer computing systems having a range of performance characteristics; the lower plot in Figure 13.10 shows all published results for the integer SPEC2006 benchmark. While hardware performance has now improved to the point where for many uses it appears to be good enough, it is still possible to buy hardware that is under-powered for the job it is expected to perform (the upper plot in Figure 13.10 shows the orders of magnitude improvements in cost and performance of sorting over 15 years).

Benchmark results published for general consumption are sometimes little more than marketing claims. The author(s) may have reasons for wanting to create a favourable impression for one system, in preference to others, or may just have done a sloppy job (perhaps because of inexperience, incompetence or lack of resources to do a decent job). One class action suite alleged that:<sup>563</sup> “Intel used its enormous resources and influence in the computing industry to, in Intel’s own words, “falsely improve” the Pentium 4’s performance scores. It secretly wrote

<sup>iii</sup> Obtaining accurate benchmark results has always been an expensive and time-consuming process, but at least it used to be possible to rely on devices sharing the same part number having the same performance characteristics.

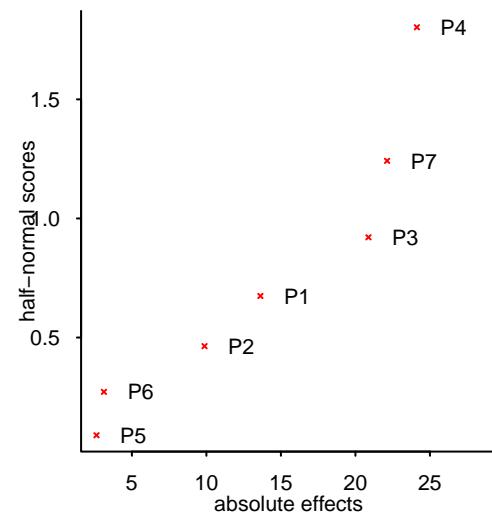


Figure 13.7: Half-normal plot of data from a Plackett and Burman design experiment. Data from Debnath et al.<sup>376</sup> code

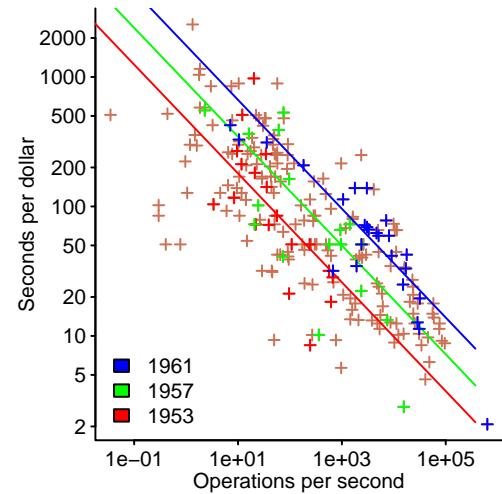


Figure 13.8: Performance and rental cost of early computers, with straight line fits for a few years. Data from Knight<sup>868</sup> code

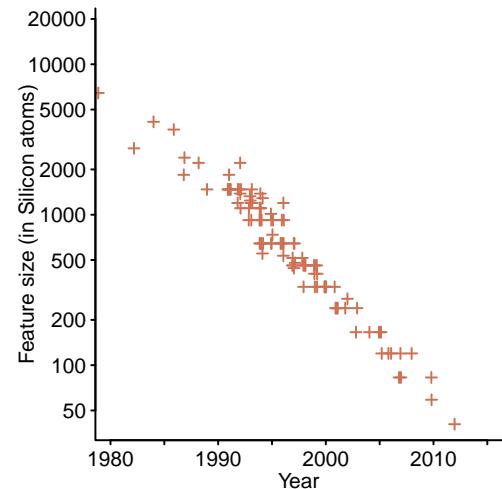


Figure 13.9: Feature size, in Silicon atoms, of microprocessors. Data from Danowitz et al.<sup>356</sup> code

benchmark tests that would give the Pentium 4 higher scores, then released and marketed these "new" benchmarks to performance reviewers as "independent third-party" benchmarks. It paid software companies to make covert programming changes to inflate the Pentium 4's performance scores and even disabled features on the Pentium III so that the Pentium 4's scores would look better by comparison.<sup>iv</sup>

In some consumer goods markets, product benchmark results receive a lot of publicity, with potential customers thought to be influenced by the results achieved by similar products. A study by Shimpi and Klug,<sup>1433</sup> of Android benchmarks, found that some mobile phone vendors detected when a particular benchmark was being run and raised the devices thermal limits (allowing the system clock rate to run faster for longer; a 4.4% performance improvement was measured).

Researchers are happy to complain about poor benchmarking practices, but are not always willing to name names.<sup>1589</sup>

In published benchmark results the Devil is in the detail, or more often in the lack of detail, as illustrated by the following:

- Bailey<sup>91</sup> lists twelve ways in which parallel supercomputer benchmarks have been written in a way likely to mislead readers, including: quoting 32-bit, not 64-bit results, quoting figures for the inner kernel of the computation, as if they applied to the complete application, and comparing sequential code against parallelized code.
- Citron<sup>297</sup> analysed the ways in which many research papers using the SPEC CPU2000 suite have produced misleading results, by only using a subset of the benchmark programs (of 115 papers surveyed, 23 used the whole suite). In one case a reported speed up of 1.42 is reduced to 1.16 when the whole suite is included in the analysis (reduction from 1.43 to 1.13 in another and from 1.76 to 1.15 in a third).

The primary purpose of this section is to highlight the many sources of variability present in modern computing systems. The available evidence suggests that large variations in benchmark results are now the norm. Large variations in measured performance do not prevent accurate results being obtained, the impact is to increase the time and money needed (i.e., it is simply a case of making enough measurements). Advice on how to perform benchmarks is available elsewhere.<sup>480,642</sup>

People interested in consistent performance will want to minimise the variation in benchmark results (which did occur for some programs), while those interested in actual benchmark performance will be interested that significant changes in the mean occurred for some programs.

When comparing different systems, benchmark performance may be normalised to produce a relative performance ranking. The geometric mean has to be used when comparing normalized values, otherwise the results can be inconsistent.

	R	M	Z	R/M	M/R	R/Z	Z/R	M/Z	Z/M
E	417.00	244.00	134.00	1.71	0.59	3.11	0.32	1.82	0.55
F	83.00	70.00	70.00	1.19	0.84	1.19	0.84	1.00	1.00
H	66.00	153.00	135.00	0.43	2.32	0.49	2.05	1.13	0.88
I	39449.00	33527.00	66000.00	1.18	0.85	0.60	1.67	0.51	1.97
K	772.00	368.00	369.00	2.10	0.48	2.09	0.48	1.00	1.00
Arithmetic	8157.40	6872.40	13341.60	1.32	1.01	1.50	1.07	1.09	1.08
Geometric	586.79	503.13	498.68	1.17	0.86	1.18	0.85	1.01	0.99

Figure 13.10: Maximum number of records sorted in 1 minute and using 1 penny's worth of system time (upper). SPEC2006 integer benchmark results (lower). Data from Gray et al<sup>618</sup> and SPEC.<sup>1471</sup> code

Table 13.1: Benchmark results for three processors (R, M, Z), running five benchmarks (E thru K), with normalisation using different processors, along with arithmetic and geometric means. Data from Fleming et al.<sup>510</sup> code

Table 13.1 shows the results of five benchmark programs (E to K) from three systems (i.e., columns R, M and Z); two other sets of columns list normalised values, with different processors used as the reference. The bottom two rows list the arithmetic and geometric mean of the columns; note: for the arithmetic mean, both ratios R/M and M/R are greater than one, while for the geometric mean one ratio is less than one (the same pattern occurs for the other ratios). A ranking based on the arithmetic mean depends on the processor used as the base for normalization, while the geometric mean produces a consistent ranking; see Section 10.3.3.

<sup>iv</sup> The class action was settled<sup>1511</sup> with Intel agreeing to pay \$15 to Pentium 4 purchasers, \$4 million to a non-profit entity and an amount not to exceed \$16.45 million to the lawyers who brought the suit.

### 13.3.1 Following the herd

When choosing a benchmark, there is a lot to be said for doing what everybody else does, advantages include:

- can significantly reduce the cost and time needed to obtain benchmark data,
- an established benchmark is likely to be usable out-of-the-box. It takes time for a benchmark to become established; an analysis<sup>1228</sup> of one Java source code corpora was only able to build 86 of the 106 Java systems in the corpus (and 56 of these had to be patched to get them to build),
- it is easier to sell the results to audiences, when the benchmark used is known to them.

The disadvantage of following the herd is that there may be fitness-for-purpose issues associated with using the benchmark, i.e., herd behavior is adapted to environments that may be substantially different from the environment in which the system is intended operate. For instance, the SPEC benchmark is often used to compare compiler performance, but SPEC's intent is for it to be used for benchmarking processor performance.

Commonly used benchmarks suffer from vendors tuning their products to perform well on the known characteristics of the benchmark. The SPEC benchmark has been used over many years for compiler benchmarking and compiler vendors often use it in-house for performance regression testing.

### 13.3.2 Variability in today's computing systems

In the good old days, computer performance tended to be relatively consistent across identical, but physically different, components, i.e., the same model of cpu or memory chip. Also, software tended to have relatively few options that could significantly alter its performance characteristics.

Modern hardware may contain components that are fabricated using handfuls of atoms, with process variations, of an atom or two here and there, producing surprisingly different performance characteristics,<sup>1094</sup> but externally looking like identical devices. Further reductions, in the number of atoms used to fabricate devices, will lead to greater variations in the final product. Today's consumer of benchmark results has to chose between:

- accepting a wide margin of error,
- executing a benchmark very many times, to ensure the sample size is large enough to achieve the desired statistical confidence interval for the results.

Both approaches require checking that a wide range of, possible unknown, factors are controlled for, by those running the benchmark.

Intrinsic variability in system performance impacts development teams that regularly monitor the performance of their products during ongoing development. For instance, Mozilla regularly measures the performance of the latest checked-in version of Firefox source code, if an update results in a performance decrease exceeding a predefined limit, the update is rolled back. Successful implementation of such a policy requires careful control of external factors that could impact performance.

Performance variation has to be addressed from a system wide perspective,<sup>v</sup> hardware/software interaction can have a significant performance impact and there are often multiple, independent, sources of variation. At the systems level differences in component characteristics (e.g., differences in system clock frequency drift in multiprocessor systems<sup>739</sup>) can interact to produce emergent effects.

DVFS (Dynamic Voltage and Frequency Scaling) provides an example of how the complexities of system component interactions make it difficult to reliably predict performance. As its name suggests, DVFS allows processor voltage and frequency to be changed during program execution; an analysis of system power consumption<sup>1693</sup> concludes that total power

---

<sup>v</sup> The following two sections separately discuss performance variation whose root cause in hardware or software; this is for simplicity of presentation.

consumed, executing a program from start to finish, is minimised by running the processor as fast as possible (assuming there is no waiting for user input).

A study by Götz, Ilsche, Cardoso, Spillner, Aßmann, Nagel and Schill<sup>604</sup> investigated how the total system power consumed by implementations of various algorithms, varied with cpu clock frequency, with the intent of finding the frequency which minimised power consumption.

Figure 13.11 shows that total power consumption does not decline with frequency, there is a frequency below the maximum that minimises power consumed.<sup>373</sup> The power minimisation frequency depends on the implementation of the sorting algorithm, and the difference between minimum and maximum depends on the number of items being sorted. Predicting the power consumed<sup>163</sup> by a program is a non-trivial problem.

Programming languages are starting to support constructs that provide developers with options for dealing with power consumption issues.<sup>1374</sup>

### 13.3.2.1 Hardware variation

This section outlines some of the evidence for large variations in hardware component performance. Much of the data used in the analysis was obtained using programs executing on components manufactured five to ten years before this book was published; variability has likely increased in the subsequent years. The hardware components covered include:

- CPU: performance, power consumption and instruction counts,
- main storage: hard disc performance and power consumption,
- memory: performance and power consumption,
- aging of components: impact on performance and power consumption.

When computing devices are connected to the mains power supply, there is rarely any need to be concerned about the characteristics of the supply. Batteries have characteristics that can affect the performance of devices connected to them, such as the level of power delivery being dependent on the current charge state and power draw frequency characteristics. In a mobile computing environment, power consumption can be just as important, if not more so, than performance. Peltonen et al<sup>1230</sup> is a public dataset of power consumption on 149,788 mobile devices, containing 2,535 different Android models; Jongerden<sup>795</sup> analyses various models of battery powered systems and Buchmann<sup>223</sup> covers rechargeable batteries in detail.

In mobile devices, a large percentage of power is consumed by the display; optimization of display intensity and choice of color,<sup>1487</sup> while an app is running, is not discussed here.

CMOS (complementary metal-oxide-semiconductor) is the dominant technology used in the fabrication of the chips contained in computing devices, and until a so-called beyond-CMOS device<sup>1159</sup> technology becomes commercially viable, only the characteristics of CMOS devices are considered.

**CPU:** Developers have generally considered the processor executing their code to be interchangeable with any of the other, mass-produced, etched slices of silicon stamped with the same model number; while never exactly true, deviations from this interchangeability assumption were once small enough to only be of interest within specialised niches, e.g., hardware modders interested in running systems beyond rated limits, such as clock rates.

The micro-architecture of modern processors has become so complicated that apparently minor changes to an instruction sequence can have a major impact on performance;<sup>738</sup> a trivial change to the source code or the use of a different compiler flag may be enough.

Power consumption and clock frequency are directly connected; increasing clock frequency increases power consumption (a good approximation for processor power consumption is  $P = \alpha FV^2 + I_0 V$ , where  $\alpha$  is a device dependent constant,  $F$  is clock frequency,  $V$  is voltage supplied to the cpu and  $I_0$  is leakage current). Processors clocked at the same frequency execute instructions at the same rate. However, variations in the number of atoms implementing internal circuitry produces variations in power consumption. Some processors reach their maximum operating temperature more quickly than others; to prevent device destruction

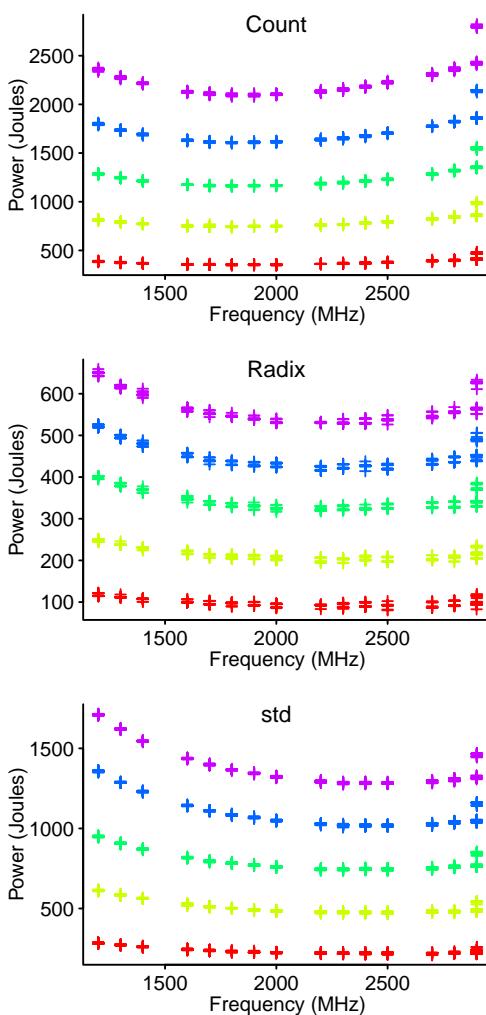


Figure 13.11: Total system power consumed when sorting 10, 20, 30, 40, 50 million integers (colored pluses) using three techniques running on the same processor at different clock frequencies. Data from Götz et al.<sup>604</sup> [code](#)

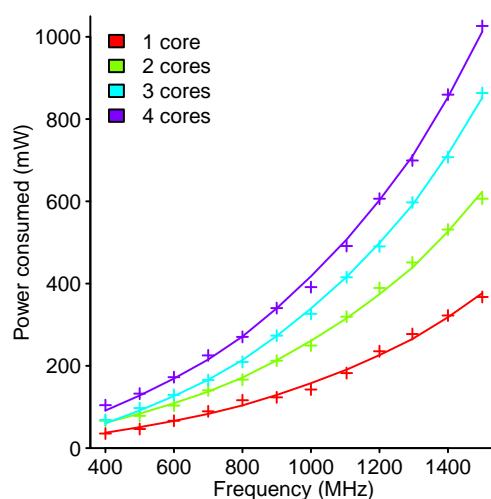


Figure 13.12: Power consumed by an Exynos-7420 A53 processor at various frequencies, and one to four cores under load, with fitted regression lines. Data kindly provided by Frumusanu.<sup>527</sup> [code](#)

through overheating, power consumption is reduced by reducing the clock rate. Different processors have different sustained performance rates because of differences in their power consumption characteristics. Vogeeler<sup>372</sup> discusses the modeling of low level temperature/power relationships for the kind of processors used to run applications.

A study by Frumusanu<sup>527</sup> measured the power and voltage, at various frequencies, of an Exynos-7420 A53 processor idling and at load. Figure 13.12 shows measured power consumption, involving one to four cores under load, at various frequencies and a fitted regression model.

Any benchmark made using a single instance of a processor is a sample drawn from a population that could vary by something like 5-10% or more when executing code and several hundred percent when idling. The extent to which results based on this minimum sample size is of practical use will depend on the questions being asked. If the power consumption characteristics of the population of a particular CPU is required, then it is necessary to benchmark a sample containing an appropriate number of *identical* processors. Methodologies for benchmarking power consumption<sup>1473</sup> require detailed attention to many issues.

A study by Wanner, Apte, Balani, Gupta and Srivastava<sup>1634</sup> measured the power consumed by 10 separate Amtel SAM3U microcontrollers at various ambient temperatures. Figure 13.13 shows a 5-to-1 difference, between supposedly identical processors, in power consumption when in sleep-mode and around 5% difference when operating at 4MHz.

Section 11.6 discusses the building of mixed-effects models for power variations involving the Intel Core processor.

How accurate are power consumption measurements? These measurements are often implemented by periodically sampling the voltage across a known resistance. A study by Saborido, Arnaoudova, Beltrame, Khomh and Antoniol<sup>1366</sup> investigated the measurement error introduced by different sampling rates, on mobile devices. Figure 13.14 shows the power spectrum of the Botanica App, executing on a BeagleBone Black running Android 4.2.2, sampled at 500K per second. By using a very high sampling rate, it is possible to see the noticeable peak in power consumed by very short-lived events, something that low frequency sampling would not detect (the paper lists error estimates for lower sampling rates).

In theory, counting the instruction executed by a program is a means of obtaining, power independent, answers to questions about comparative program performance. Some processors include hardware support for counting the number of operations performed, e.g., instruction opcodes executed and cache misses; however, the purpose of these counters is to help manufacturers debug their processors, not provide end-user functionality. Consequently, counter values are not guaranteed to be consistent across variants of processors within the same family<sup>1643</sup> and fixing faults in the counting hardware does not have a high priority (e.g., counting some instructions twice or not at all; Weaver and Dongarra<sup>1643</sup> found that in most cases the differences were a fraction of a percent of the total count, but for some kinds of instructions, such as floating-point, the counts were substantially different). A study by Weaver and McKee<sup>1644</sup> found that it was possible to adjust for the known faults in the hardware counters; see [benchmark/iswc2008-i686.R](#).

How are calls into operating system routines, which may execute at higher privilege levels, counted? Also, the execution time of some instructions may depend on other instructions executed at around the same time (modern processors have multiple functional units and allocate resources based on the instructions currently in the pipeline), the time taken to execute other instructions can be very unpredictable (e.g., time taken to load a value from memory depends on the current contents of the cache and other outstanding load requests). A study by Melhus and Jensen<sup>1062</sup> showed that address aliasing, of objects in memory, could have a huge impact on the relative values of some hardware performance counters.

Hardware counters need not be immune to *observer effects*; in particular, the values returned can depend on the number of different hardware counters being collected.<sup>1126</sup>

**Main storage:** Traditionally main storage has meant hard disks (sometimes backed-up with tape), but solid state devices (SSD) are rapidly growing in capacity<sup>1524</sup> and use; for extremely large capacity magnetic tape is used: this niche use is not discussed here.

Data is read/written to a hard disk by moving magnetic sensors across a rotating surface. These spatial movements create a correlation between successive operations, e.g., the time taken to perform the second read will depend on its location on the disk relative to the first. Disk access issues have been studied for over 60 years and techniques such as data buffering,

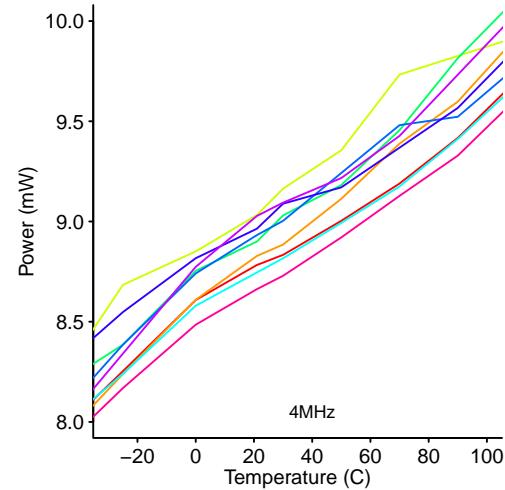
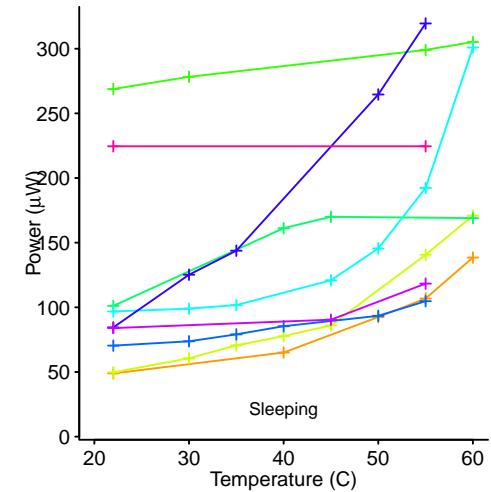


Figure 13.13: Power consumed by 10 Amtel SAM3U microcontrollers at various temperatures when sleeping or running. Data from Wanner et al.<sup>1634</sup> [code](#)

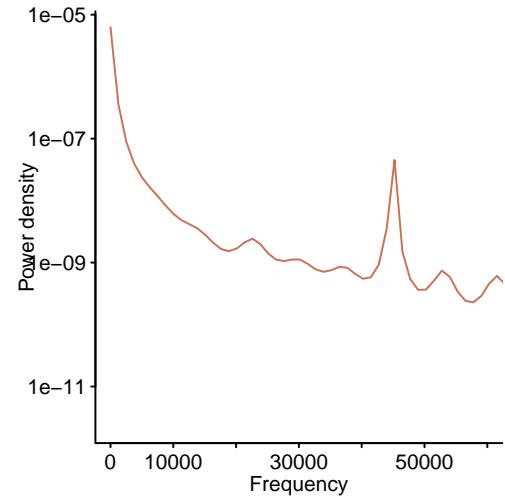


Figure 13.14: Power spectrum of electrical power consumed by the Botanica App executing on a BeagleBone Black running Android 4.2.2. Data from Saborido et al.<sup>1366</sup> [code](#)

by the operating system, and reordering requests to optimise overall throughput, by the device driver or firmware, are well-established.<sup>209</sup>

Disk spin at a constant rate, and in the same time interval more data can be read from the area swept out near the outer edge of a platter than from one near the spindle (the staircase effect in the upper plot in Figure 13.15 is a result of zoning). This location dependent performance characteristic makes disk benchmark performance dependent on the history of the data that has been added/deleted.

Storage farms organise files so that those most likely to be accessed are stored on the outer tracks, while files less likely to be accessed are stored on the inner tracks. A growing percentage of disks are used in data centers and at some point manufacturers may decide to concentrate on designing drives for this market.<sup>209</sup>

The continuing increase in the number of bits that can be stored within the same area of rotating rust has been achieved by reducing the size of the magnetic domain used to store a bit. Like silicon wafer production, variations in the fabrication process of disc platters can now result in large differences in the performance of supposedly identical drives.

A study by Krevat, Tucek and Ganger<sup>884</sup> measured the performance of disk drives originally sold in 2002, 2006, 2008 and 2009. In Figure 13.15, the upper plot shows the read bandwidth of nine disks from 2002, each displayed using a different color and there is little variation between different disks (fitting a regression model finds that disk identity is not a significant predictor of performance, p-values around 0.2). The lower plot shows the read bandwidth of nine disks from 2006, each displayed using a different color; the visibility of different colors shows the variation between different disks (fitting a regression models finds that disk identity is a significant component of performance prediction, p-values around  $10^{-16}$ ).

To increase recording density, drive manufacturers are now using Shingled Magnetic Recording (SMR), where tracks overlap like rows of shingles on a roof. Singled discs have very different performance characteristics,<sup>12</sup> but little data is publicly available at the time of writing.

SSDs are sufficiently new that little performance data is publicly available at the time of writing.

A study by Kim<sup>843</sup> ran eight different benchmarks on SSD cards from nine different vendors. The range of performance values was different for both vendors and benchmark. Building a regression model, using normalised benchmark scores, finds that one vendor's products have a sufficiently consistent performance that they can be included in a model (these products appear to have the best performance, the other vendors appear to have the same performance); see [benchmark/hyojun/hyojun.R](#).

**Memory:** Memory chips tend to be thought about in terms of their capacity and not their performance (such as, read/write delays or power consumption). Performance is governed by access rate and by the number of bytes transferred per access, with accesses usually made via some form of memory control chip (the capabilities of this controller have a significant impact on performance). Many motherboards provide options to select memory chip timing characteristics.

A study by Bircher<sup>163</sup> investigated the power consumed by the various hardware of a server, while it executed the SPEC CPU2006 benchmark. The upper plot in Figure 13.16 breaks down average power by CPU (red) and memory (blue), while the lower plot breaks the power down by the major subcomponents of the server.

There is often a performance hierarchy for memory, with on (cpu) chip cache providing faster access to frequently used data. The interaction between the size of the various memory caches and an algorithm's use of storage can result in performance characteristics that change as the size of the objects processed grows.

A study by Khuong and Morin<sup>840</sup> measured the performance of several search algorithms, when operating on arrays of various sizes (items were stored appropriately in the array, for the algorithm used). Figure 13.17 shows the time taken by different algorithms to find an item in an array, for arrays of various sizes; the grey lines show the total size of the processor L1, L2 and L3 caches.

The following are some examples of memory chip characteristics that have been found to noticeably fluctuate:

- a study by Gottscho, Kagalwalla and Gupta<sup>603</sup> measured power consumption variability of 13 DIMMs, of the same model of 1G DRAM from four vendors. The variation about

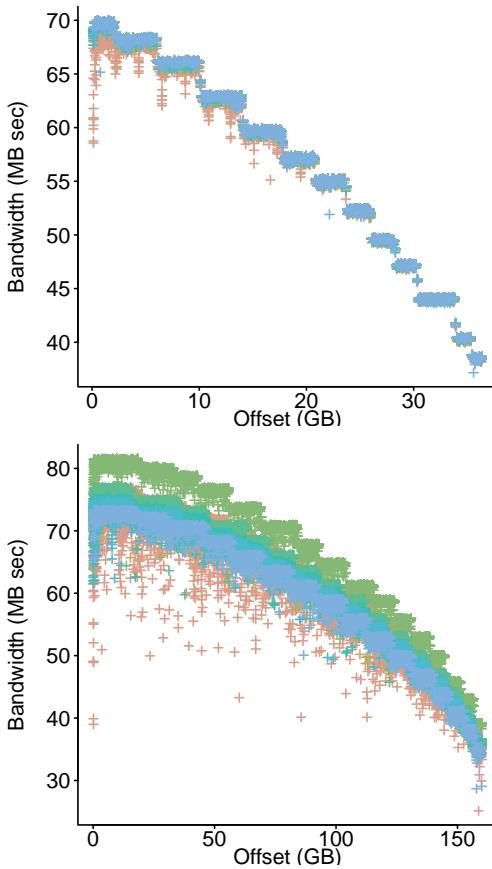


Figure 13.15: Read bandwidth at various offsets for new disks sold in 2002 (upper) and 2006 (lower). Data kindly provided by Krevat.<sup>884</sup> [code](#)

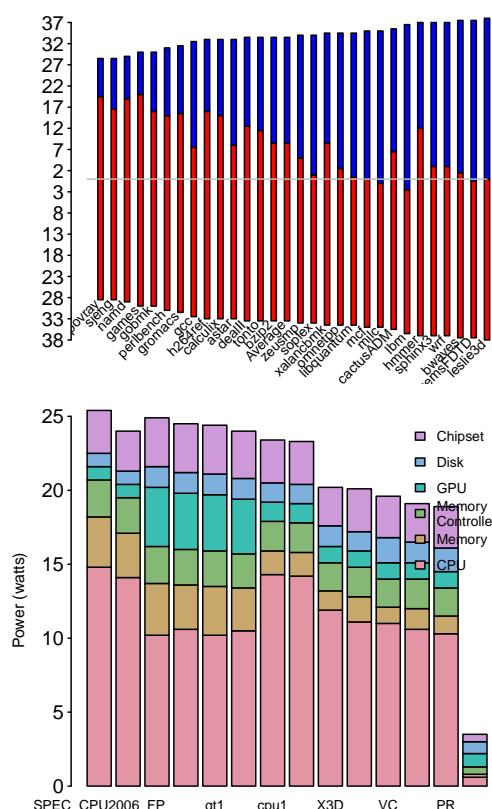


Figure 13.16: Average power consumed by one server's CPU (four Pentium 4 Xeons; red) and memory (8 GB PC133 DIMMs; blue) running the SPEC CPU2006 benchmark (upper) and breakdown by system component when executing various programs. Data from Bircher.<sup>163</sup> [code](#)

the mean, at one standard deviation, was 5% for read operations, 9% for write and 7% for idling; see [benchmark/J20\\_paper.R](#):

- a study by Gottscho<sup>602</sup> measured the power consumption of 22 DDR3 DRAMs, manufactured in 2010 and 2011, from four vendors. Read operations consumed around 60% of the power needed for write operations, with idle consuming around 40%; the standard deviation varied from 10% to 20%. The power consumed also varied with value being read/written, e.g., writing 1 to storage containing a 0 required 25% more power than writing a 0 over a 1; see [benchmark/MSTR10-DIMM.R](#) for data.
- a study by Schöne, Hackenberg and Molka<sup>1389</sup> found that memory bandwidth was reduced by up to 60%, as the frequency of the cpu was reduced, that memory performance characteristics varied between consecutive generations of Intel processors and between server and desktop parts,

The variability of memory chip performance is likely to increase, as vendors further reduce power consumption and improve performance by lengthening DRAM refresh times; optimising each computer by tuning it to the unique characteristics of the particular chips present in each system.<sup>935</sup>

Chandrasekar<sup>267</sup> provides a detailed discussion of DRAM power issues, including code for a tool to obtain detailed information about the memory chips installed on a system.

**Aging of components:** emailed for data...

### 13.3.2.2 Software variation

This section outlines some evidence for large variations in software performance, briefly covering the following software components and processes:

- The environment: interaction with the environment, file system, support libraries and aging,
- Configurations,
- Creating an executable: compiler optimization and link order,
- Tools.

**The environment:** Programs execute within an environment that often contains a complicated ensemble of interconnecting processes and services that cannot be treated as independent standalone components. One consequence of this complexity,<sup>782</sup> and interconnectedness, is that the order in which processes are initiated during system startup can have a noticeable impact on system performance.

The impact of a system's prior history, on program performance, is seen in a study by Kalibera, Bulej and Tůma,<sup>812</sup> who measured the execution time of multiple runs of various programs. Figure 13.18 shows 10 iterations of the procedure: reboot computer and make 2,048 performance measurements. The results show performance variation after each reboot is around 0.1%, but rebooting can cause a shift of 3% in the average performance (the ordering of processes executed during system startup varies across reboots, due to small changes in the time taken to execute the many small scripts that are invoked during startup; execution ordering effects placement of data in memory, which can have an impact on performance). A later study<sup>707</sup> found that the non-determinism of initial program execution, in this case, could be reduced by having the operating system use cache-aware page allocation.

Environmental interactions are not always obvious. A study by Mytkowicz, Diwan, Hauswirth and Sweeney<sup>1126</sup> increased the number of bytes occupied by a Linux environment variable between runs of the perlbench program. The results from each of 15 executions were recorded, an environment variable increased in size by one character and the procedure repeated 100 times. Figure 13.19 shows the percentage change in performance, relative to the environment variable containing zero characters, at each size of environment variable, along with 95% confidence intervals of the mean of each 15 runs.

Incremental operating system updates can produce a change in program performance. A study by Flater<sup>508</sup> compared the performance of cpu intensive and I/O bound programs on two

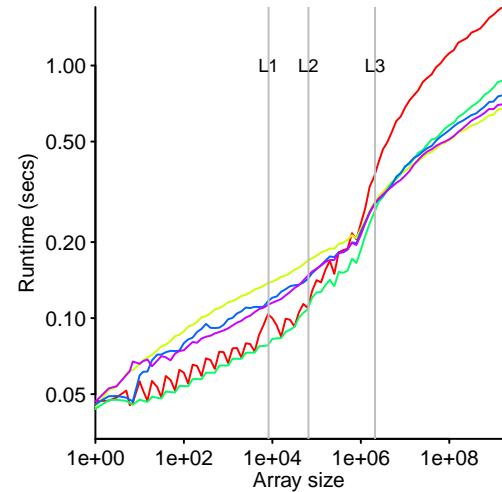


Figure 13.17: Time taken to find a unique item in arrays of various size, containing distinct items, using various search algorithms; grey lines are L1, L2 and L3 processor cache sizes. Data from Khuong et al.<sup>840</sup> [code](#)

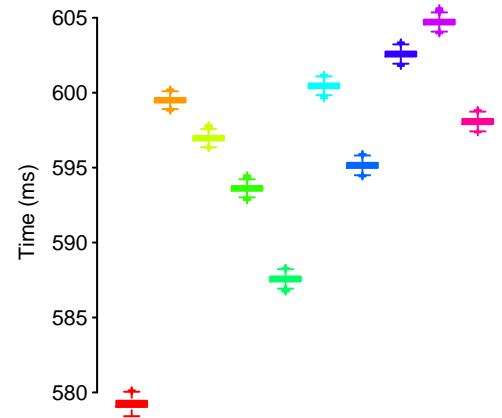


Figure 13.18: FFT benchmark executed 2,048 times followed by system reboot, repeated 10 times. Data kindly provided by from.<sup>812</sup> [code](#)

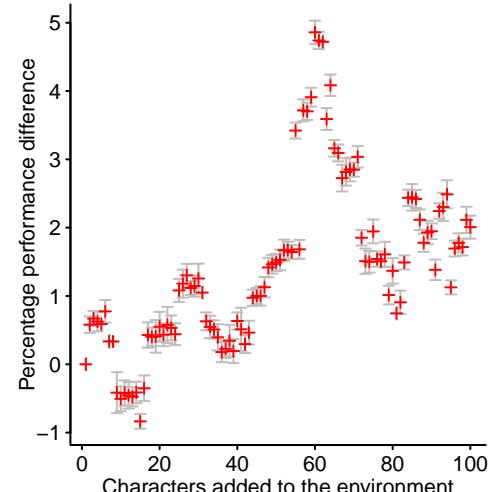


Figure 13.19: Percentage change, relative to no environment variables, in perlbench performance as characters are added to the environment. Data extracted from Mytkowicz et al.<sup>1126</sup> [code](#)

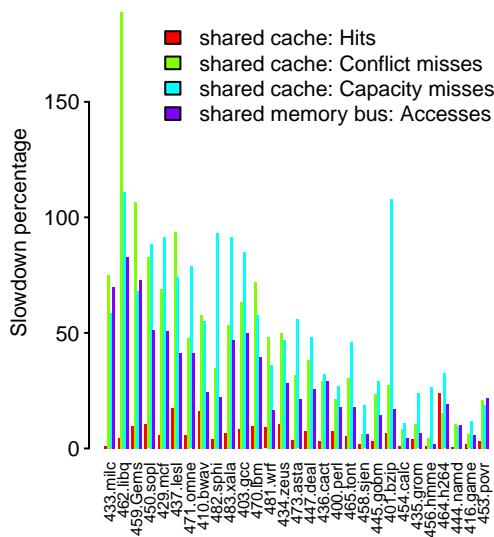


Figure 13.20: Changes in SPEC CPU2006 performance caused by cache and memory bus contention, for one dual processor Intel Xeon E5345 system. Data kindly provided by Babka.<sup>77</sup> code

different versions of Slackware, running on the same hardware (versions 14.0 and 14.1, using Linux kernels 3.12.6 and 3.14.3 respectively). The results show consistent differences in performances of up to 1.5% (rebooting did not have any significant impact on performance).

Many systems allow multiple programs to be executing at the same time, sharing system resources. Sharing becomes a performance bottleneck when one program cannot immediately access resources when it requests them; access to memory is a common resource contention issue on multi-processing systems. A study by Babka<sup>77</sup> investigated the performance of multicore processors having a shared cache. Figure 13.20 shows changes in SPEC CPU2006 performance caused by cache and memory bus resource contention, on a dual processor Intel Xeon E5345 system.

A study by Mazouz<sup>1031</sup> investigated the performance of the SPEC OpenMP 2001 programs, compiled using gcc 4.3.2 and icc 11.0, running on multicore devices. It is possible for a program's code to execute on a different core after every context switch. Allowing the operating system to select the core to continue program execution is good for system level load balancing, but can reduce the performance of individual programs because recently accessed data is less likely to be present in the cache of any newly selected core. *Thread affinity* is the process of assigning each thread to a subset of cores, with the intent of improving data locality, i.e., recently accessed data is more likely to be available in accessible caches.

Figure 13.21 shows the time taken to execute one program in 2, 4, and 6 threads, with thread affinity set to compact (threads share an L2 cache), no affinity (allow the OS to assign threads to cores) and scatter (distribute the threads evenly over all cores), each repeated 35 times.

Configuring the system being benchmarked to only run one program at a time solves some, but not all, cache contention issues. Walking through memory, in a loop, may result in a small subset of the available cache storage being used (main memory is mapped to a much smaller cache memory, which means that many main memory addresses are mapped to the same cache address). Figure 13.22, from a study by Babka and Tůma,<sup>78</sup> shows the effect of walking through memory using three different fixed width strides; for 32 and 64 byte strides accesses to even cache lines is faster than odd lines, with the pattern reversed for a 128 byte stride.

Operating systems generally have background processes that spend most of the time idling, but wake up every now and again. When a background processes wakes up, it will consume system resources and can have an impact on the performance reported by a benchmark, i.e., background processes are a source of variation.

A study by Larres<sup>922</sup> investigated how the performance of one version of Firefox changed as various operating system features were disabled (the intent being to reduce the likelihood that external factors added noise to the result). The operating system features modified were: 1) every process that was not necessary was terminated, 2) address-space randomization was disabled, 3) the Firefox process was bound exclusively to one cpu, and 4) the Firefox binary was copied to and executed from a RAMDISK.

Every program in the Talos benchmark (the performance testing framework used by Mozilla) was run 30 times. Figure 13.23 shows the performance of various programs running in original and stabilised (i.e., low-noise) configurations.

**File systems:** These are a way of organising information on a storage device. The traditional view of a file, as a leaf in a directory tree, has become blurred, with many file system managers now treating compressed archived files (e.g., zip files) as-if they had a directory structure that can be traversed; Microsoft's .doc format contains a FAT (File Allocation Table, just like a mounted Windows file system) that can refer to contents that may exist outside of the file.

A study by Zhou, Huang, Li and Wang<sup>1711</sup> investigated the performance interplay between file systems and Solid State Disks (SSD), by running a file-server benchmark on a Kingston MLC 60GB SSD. Four commonly used Linux filesystems (ext2, ext3, reiserfs and xfs) were mounted in turn using various options, e.g., various block sizes, noatime, etc.

Figure 13.24 shows the number of operations per second for a file-server benchmark (see paper and data for other benchmarks). The data is poorly fitted by a linear regression model (i.e., not significant on the filesystem or mount options; see [benchmark/filesystem-SSD.R](#)).

A study by Sehgal, Tarasov and Zadok<sup>1405</sup> compared the power used when four commonly used filesystems were mounted in various ways, e.g., fixed vs. variable sector size, different journal modes, etc. Various server workloads running on Linux were measured; web server

power consumption varied by a factor of eight, mail server by a factor of six and file and database by a factor of two.

**Creating an executable:** Many applications are built by translating source code to an executable binary, with the translation tools often supporting many options, e.g., gcc supports over 160 different options for controlling machine independent optimization behavior. Compiler writers strive to improve the quality of generated code and it is to be expected that the performance of each release of a compiler will be different from the previous one; there have been around 150 released versions of gcc in its 30 year history.

A study by Makarow<sup>1011</sup> measured the performance of nine releases of gcc, made between 2003 and 2010, on the same computer using the same benchmark suite (SPEC2000), at optimization levels 02 and 03.

Figure 13.25 shows the percentage change in SPEC number, relative to version 4.0.4, for the 12 integer benchmark programs compiled using six different versions of gcc. SPEC has a long history of being used for compiler benchmarking, and it is possible that the versions of gcc used for this comparison have already been tuned to do well on this benchmark, meaning there is little, benchmark specific, improvement to be had in the successive versions used in this study.

The following summary output is from a mixed-effect model with the random effect on the intercept and slope: [code](#)

```
Linear mixed model fit by REML [‘lmerMod’]
Formula: value ~ gcc_version + (gcc_version | Name)
Data: lme_02
```

REML criterion at convergence: 400.6

Scaled residuals:

Min	1Q	Median	3Q	Max
-2.7256	-0.2748	-0.0683	0.3039	4.3372

Random effects:

Groups	Name	Variance	Std.Dev.	Corr
Name	(Intercept)	1192.795	34.537	
	gcc_version	3.155	1.776	-1.00
Residual		8.632	2.938	

Number of obs: 72, groups: Name, 12

Fixed effects:

	Estimate	Std. Error	t value
(Intercept)	-29.7469	11.0553	-2.691
gcc_version	1.4126	0.5513	2.562

Correlation of Fixed Effects:

(Intr)	gcc_version
-0.997	

The general picture painted by the model results is of a small improvement with each gcc release, which is swamped by the size of the random effects, while the picture painted by Figure 13.25, is of some releases having a large impact on some programs.

A study by de Oliveira, Petkovich, Reidemeister and Fischmeister<sup>370</sup> investigated the impact of compiler optimization and object module link order on program performance. Figure 13.26 shows the time taken by the xy file compression program, compiled by gcc using various optimization options, to process the Maximum Compression test set on various systems. The results show that different optimization levels have a different performance impact on different systems (the lines would be parallel if optimization level had the same impact for each system).

Compiling is the first step in the chain of introducing system variability into program performance, the next step is linking. Figure 13.27 shows execution time of Perlbench (one of the SPEC benchmark programs), on six systems, when the object files used to build the executable are linked in three different orders and with address randomization on/off. Some systems share a consistent performance pattern across link orderings and some systems are not affected by address randomization. But there is plenty of variation across all the variables measured.

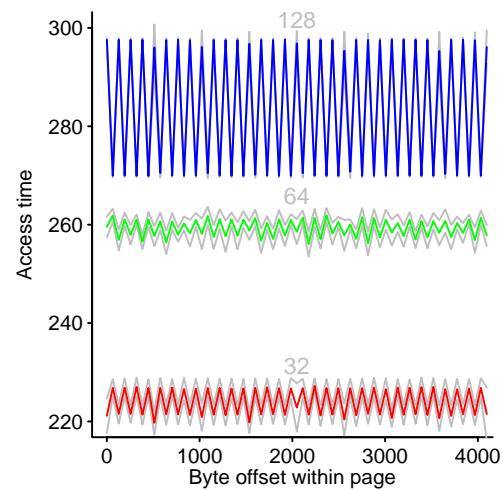


Figure 13.22: Access times when walking through memory using three fixed stride patterns (i.e., 32, 64 and 128 bytes) on a quad-core Intel Xeon E5345; grey lines at one standard deviation. Data kindly provided by Babka.<sup>78</sup> [code](#)

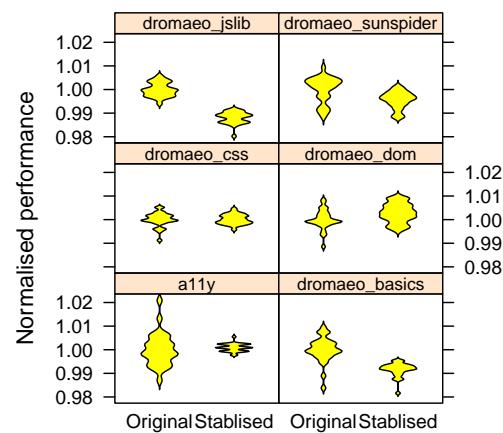


Figure 13.23: Performance variation of programs from the Talos benchmark run on original OS and a stabilised OS. Data from Larres.<sup>922</sup> [code](#)

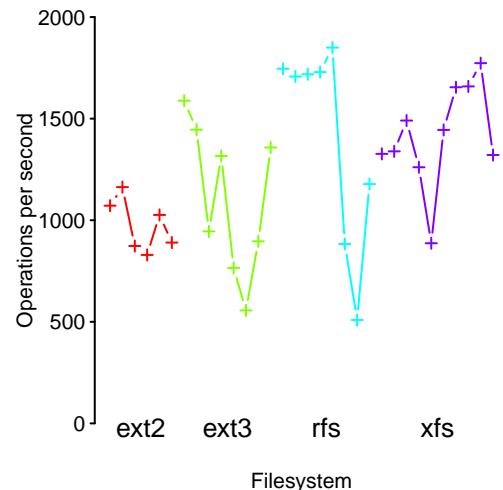


Figure 13.24: Operations per second of a file-server mounted on one of ext2, ext3, rfs and xfs filesystems (same color for each filesystem) using various options. Data kindly supplied by Huang.<sup>1711</sup> [code](#)

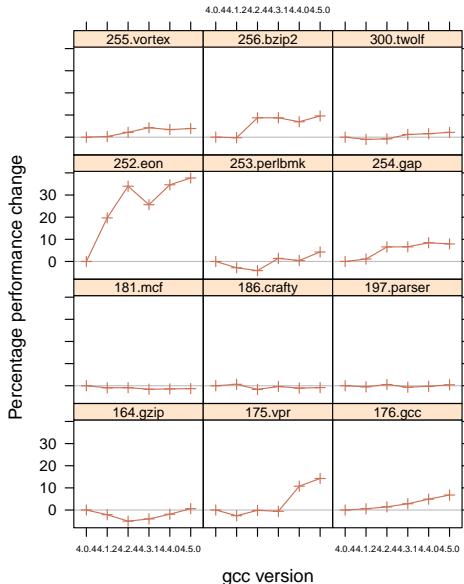


Figure 13.25: Percentage change in SPEC number, relative to version 4.0.4, for 12 programs compiled using six different versions of gcc (compiling to 64-bits with the `-O3` option). Data from Makarow.<sup>1011</sup> [code](#)

**Tools:** Dynamic profiling tools such as `grprof` work by interrupting a program at regular intervals during execution (e.g., once every 0.01 seconds) and recording the current code location (often at the granularity of a complete function). The results obtained can depend on interrupt frequency and the likelihood of being in the process of calling/returning from the profiled function.<sup>507</sup>

### 13.3.3 End user systems

Benchmark data supplied by end-users, run on the computing systems they own, is likely to be subject to numerous known and unknown unknowns.

PassMark Software specializes in benchmark solutions<sup>1680</sup> and over the years has collected 800,000+ benchmark results from Microsoft Windows based computers; Wren<sup>1680</sup> kindly provided 10,000 memory benchmark results. Background processes that may be running on a Windows machine include: Internet based toolbars, anti-virus systems and general OS housekeeping processes.

Figure 13.28 shows the results (in sorted order) from 783 systems containing an Intel Core i7-3770K processor (whose official clock speed is 3.5GHz, some users may be overclocking). This is another example (see Figure 10.24) of the wide range of performance reported for apparently very similar end-user systems.

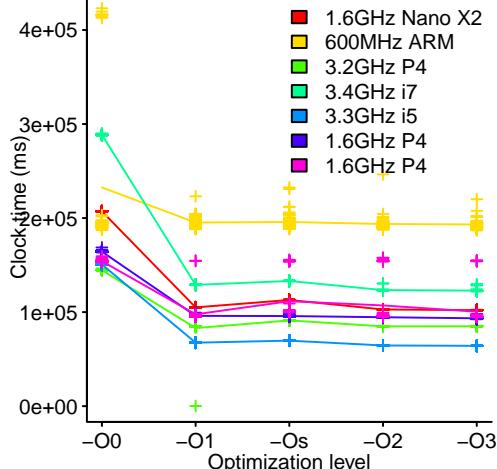


Figure 13.26: Execution time of `xy` file compressor, compiled using gcc using various optimization options, running on various systems (lines are mean execution time when compiled using each option). Data kindly supplied by Petkovich.<sup>370</sup> [code](#)

### 13.3.4 The cloud

Cloud computing has become a popular platform for applications that require non-trivial compute resources. The service level agreements offered by cloud providers specify minimum levels of service, e.g., Amazon's June 2013 EC2 terms specify 99.95% monthly uptime.<sup>35</sup> Cloud services generally run virtualized instances, which means access to the real hardware may sometimes be shared. Shared hardware access causes performance to vary from one run to the next; what form might the characteristics of this variation take?

A study by Schad, Dittrich and Quiané-Ruiz<sup>1380</sup> submitted various benchmarks, as jobs, to Amazon's Elastic Computing Cloud (EC2), twice an hour over a 31-day period; a variety of resource usage measurements were recorded. Figure 13.29 shows one set of resource usage measurements, the Unix benchmark utility (`Ubench`; a cpu benchmark) running on small (upper) and large (lower) EC2 instances located both in Europe (red) and the US (green).

Both plots show more than one distinct ranges of performance. This data is an example of the variation experienced in Amazon's EC2 performance over one particular time period, and there is no reason to believe that any subsequent benchmarking will exhibit one, two, three or more distinct performance ranges.

## 13.4 User interface testing

When software systems being built by a small group of people, software developers may be called on to solve a wide range of computer related issues, including the testing and tuning of the user interface.

The System Usability Scale (SUS)<sup>216,217</sup> is a widely used usability questionnaire that produces a single number for usability. One study<sup>1569</sup> compared five methods of evaluating website usability, and found that SUS produced the most consistent results for smaller sample sizes.

User interface design... some data...

## 13.5 Surveys

This section discusses questionnaire surveys. Organizations use surveys are used to obtain information about customers and the market(s) they are targeting, e.g., characteristics of open source developers,<sup>1342</sup> see Figure 8.13. Applications need to run reasonably well on the computers that customers currently use (see Figure 8.27), and to coexist (or interoperate)

with the versions of libraries and other applications installed on these computers. A lot of software engineering information only exists in the heads' of the people who build software systems, and this information can only be obtained by asking these people questions and analysing their answers.

The survey package supports the analysis of samples obtained via surveys.

Some of the characteristics of data encountered in survey samples, include:

- missing data: people don't answer all the questions or stop answering after some point,
- misleading answers: giving answers that show those involved in a better light, such as job adverts listing trendy topics and languages to attract more applicants,
- spatial information: how subjects are distributed geographically,

Studies have found<sup>421</sup> that self-assessment of skills and character have a tenuous to modest relationship with actual performance and behavior. The correlation between self-ratings of skill and actual performance in many domains is moderate to meager.

Several studies by your author<sup>786, 787, 789</sup> included a component that asked developers about how many lines they had read and written during their professional career.

This question requires a lot of thought to answer and there are many ways of adding up the numbers. Does reading the same line twice count as two lines, or one line unless the developer involved had forgotten reading it? How much does visually searching a screen of code for a particular identifier count towards lines read? Counting the number of lines in the programs written by a developer is likely to underestimate the number of lines they have written; a line of code may be written and then deleted, an existing line may be modified slightly.

Figure 13.30 shows the number of lines of code that 101 professional developers estimate they have written. While an exponential model may fit the data, the variance explained is small...

A survey of the knowledge, or skill, of members of a population requires subjects to provide correct answers to questions.

What is the probability that a subject,  $m$ , will give the correct answer to the  $i^{th}$  question,  $x_{mi}$ , when the subject has a knowledge/skill level of  $z_m$ ?

The answer given by Item response theory (IRT)<sup>124</sup> is:

$$P(x_{mi} = 1 | z_m) = c_i + (1 - c_i)g(\alpha_i \times (z_m - \beta_m))$$

where:  $c_i$  is the probability that a subject will guess the correct answer,  $\alpha_i$  is a measure of how well the question discriminates between subjects having a low/high level,  $\beta_i$  the question difficulty, and  $g(.)$  a link function (often the logit function; the default used by the function in the `ltm` package).

The Rasch model is simpler, and widely used; it contains a single parameter,  $\beta_i$ , and assumes there is no guessing (i.e.,  $c_i = 0$ ) and questions share the same ability to discriminate between subjects at different levels (i.e.,  $\alpha_i = 1$ ); the logit function is used as the link function, and the equation is:

$$P(x_{mi} = 1 | z_m) = \frac{e^{z_m - \beta_m}}{1 + e^{z_m - \beta_m}}$$

The `ltm` package supports the of fitting Item response models having one (the Rasch model), two ( $\alpha_i$  is included) and three (all parameters are included) parameter models. The logit function is the default link function used.

A study by Dietrich, Jezek and Brada<sup>404</sup> investigated one aspect of developer knowledge of a language: knowledge of Java type compatibility. The yes/no answers to 22 questions provided by the 184 professional developers can be fitted to an IRT model.

The following code fits a Rasch model (single parameter), a two parameter model using the `ltm` function (`z1` and `z2` are dummy names used to denote each factor). A three parameter model fails to be fitted by the `tpm` function.

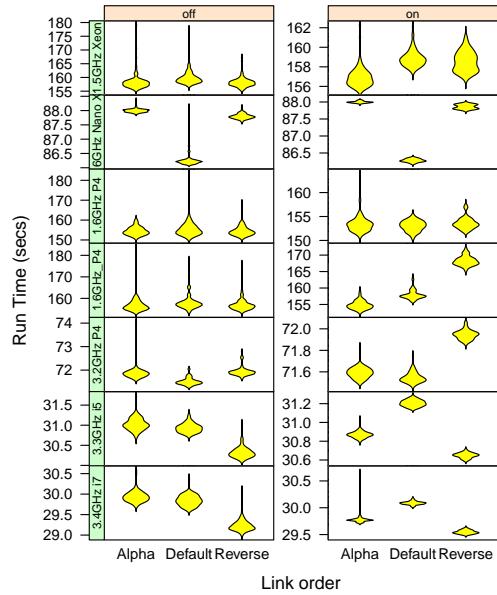


Figure 13.27: Execution time of Perlbench, from SPEC benchmark, on six systems, when linked in three different orders and address randomization on/off. Data kindly supplied by Reidemeister.<sup>370</sup> [code](#)

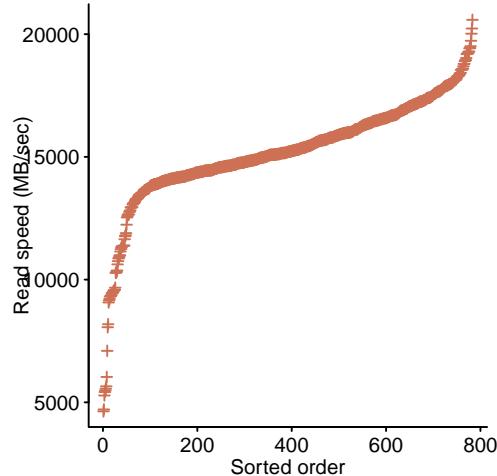


Figure 13.28: Performance of PassMark memory benchmark on 783 Intel Core i7-3770K systems. Data kindly supplied by Wren.<sup>1680</sup> [code](#)

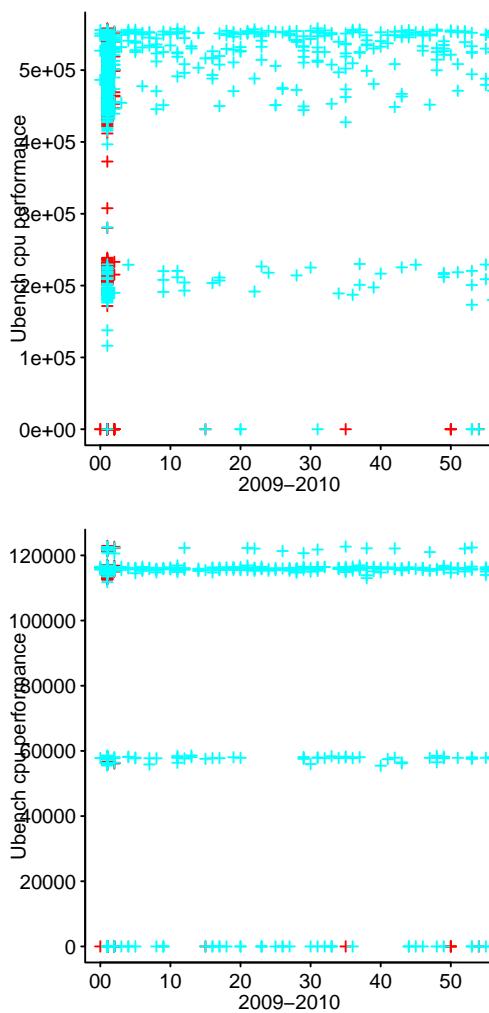


Figure 13.31 shows: the probability that a subject having a given level of ability will correctly answer each question.

Section 12.4.2 discusses sorting items in a preferred ordering...

The results of many studies<sup>47</sup> have found that most subject ratings are based on an ordinal scale (i.e., there is no fixed relationship between the difference between a rating of 2 and 3 and a rating of 3 and 4), that some subjects will be overly generous or miserly in their rating, and that without strict rating guidelines different subjects apply different criteria when making their judgements (which can result a subject providing a list of ratings that is inconsistent with all other subjects).

145 Questions for data scientists in software engineering<sup>130</sup>

Sequence mining of development techniques used in same company?<sup>?, ?, ?</sup>

2008 answers:<sup>?</sup> 2009 answers:<sup>?</sup>

Figure 13.29: Ubench cpu performance on small (upper) and large (lower) EC2 instances, Europe in red and US in green. Data kindly provided by Dittrich.<sup>1380</sup> code

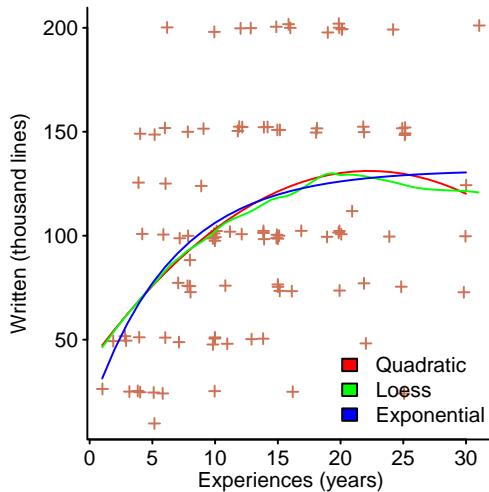


Figure 13.30: Number of lines of code that 101 professional developers, with a given number of years experience, estimate they have written, lines are various regression fits. Data from Jones.<sup>786, 787, 789</sup> code

# Chapter 14

## Data preparation

### 14.1 Introduction

The most important question to keep asking yourself while examining, preparing and analyzing any data is: Do I believe this data?

Do patterns appear where none are expected, are expected patterns absent, are human errors missing from the raw data, does the data collector believe whatever they are told, does the measurement process create incentives for people to game it?

Books and presentations on data analysis rarely mention that, often a significant amount of effort has to be invested in data preparation (perhaps 80%, or more, of analysis effort), getting it into a form that can be input to the chosen statistical analysis techniques.<sup>i</sup>

Perhaps the largest task within data preparation is data cleaning; an often overlooked<sup>957</sup> aspect of data analysis that is an essential part of the workflow needed to avoid falling foul of the adage: garbage in, garbage out.

Domain knowledge is essential for data cleaning; patterns have to be understood in the context in which they occur. The fact that many data cleaning activities are generic does not detract from the importance of domain knowledge. For instance, software knowledge tells us that 1.1 is not a sensible measurement value for lines of code (this appears in the NASA MDP dataset), talking to developers at a company to discover they don't work at weekends (e.g., the dates in the 7digital data) and knowing that system support staff used the Unix `pwd` command to check that the system was operational (an analysis of job characteristics for a NASA supercomputer<sup>481</sup> has to first remove 56.8% of all logged jobs, which are uses of `pwd`).

A study by Cohen, Teleki and Brown<sup>311</sup> investigated data from 2,751 code reviews, from one company over a 10-month period. During the data cleaning process they removed all code reviews reported taking less than 30 seconds, involving more than 2,000 LOC and processing code at a rate greater than 1,500 lines per hour. Figure 14.1 shows all reported data points, with points inside the triangle being the only measurements retained for analysis.

Data cleaning is often talked about as-if it is something that happens before data analysis, in practice, the two activities intermingle; the time spent checking and cleaning the data provides insights that lead to a better understanding of the kinds of analysis that might be applicable, also, results from a preliminary analysis can highlight data needing to be cleaned in some way. Data preparation is discussed here in its own chapter, as-if it was performed as a stand-alone activity, in order to simplify the discussion of the material in other chapters.

Once cleaned, data may need to be restructured, e.g., rows/columns contained in different files merged into a single data table, or the row/columns in an existing table reorganised in some way. The required structure of the data is driven by the operations that need to be performed on it (e.g., finding the median value of some attribute), or the requirements of the library function used to perform the analysis.

It may be necessary to remove confidential information from the data, or to anonymize information that might be used to identify individuals (or companies). Datasets do not exist

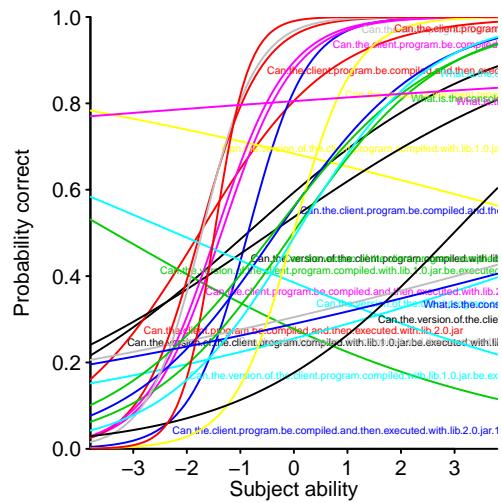


Figure 13.31: Probability that a subject, having a given relative ability, will answer a question correctly: lines are for each question in a fitted Rasch model. Data from Dietrich et al.<sup>404</sup> code

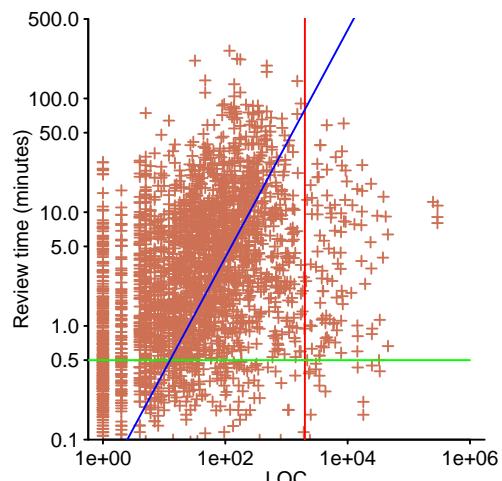


Figure 14.1: Reported LOC and duration of 2,751 code reviews, from one company; reported reviews lasting less than 30 seconds (below green line), involving more than 2,000 LOC (to right of red line), processing at a rate greater than 1,500 LOC per hour (above blue line). Data extracted from Cohen et al.<sup>311</sup> code

<sup>i</sup> In early versions, this chapter appeared immediately after the Introduction, but in response to customer demand, it was moved here (the penultimate chapter); people want to read about the glamorous stuff, data analysis, not the grunt work of data preparation.

in isolation, and it may be possible to combine apparently anonymous datasets to reveal information;<sup>ii</sup>  $k$ -anonymity and  $l$ -diversity are popular techniques for handling anonymity requirements (in a  $k$ -anonymized dataset each record is indistinguishable from at least  $k - 1$  other records, while  $l$ -diversity requires at least  $l$  distinct values for each sensitive attribute). Techniques for anonymizing data are not covered here; Fung et al<sup>530</sup> survey techniques for privacy-preserving data publishing, Templ et al<sup>1542</sup> provide an introduction to statistical disclosure control and the `sdcMicro` package.

While a lot of software engineering data comes from measurements made by programs, some is still sourced from human written records (which can contain a substantial number of small mistakes<sup>779</sup>).

Data cleaning involves a lot of grunt work that often requires making messy trade-offs and having to make do. Tools are available to reduce the amount of manual work involved, but these sometimes require placing trust in the tool doing the right thing; some of the available tools include:

- OpenRefine<sup>1192</sup> (was Google Refine) reads data into a spreadsheet-like form and supports sophisticated search/replace and data transformations editing,
- the `editrules` package checks that data values are consistent with user specified consistency rules involving named columns (e.g., `total_fruit == total_apples+total_oranges`),
- the `deducecorrect` package performs automatic value transformation based on user specified consistency rules, relating to column values that must be met (e.g., failure to meet the condition `total_x > 0` will result in any value in that column having a negative sign removed); this package can also impute missing values,
- there are a variety of special purpose packages that handle domain specific data, e.g., the `CopyDetect` package detects copying of exam answers in multi-choice questions.

### 14.1.1 Documenting cleaning operations

Documenting the changes made to the original data, during the cleaning process, serves a variety of purposes, including:

- enabling third-parties to check that the changes are reasonable and don't produce an unrealistic analysis,
- enabling potential sources of uncertainty to be checked when multiple analysts publish results based on the same dataset, i.e., if there are differences in the results, it is possible to check whether these differences are primarily the result of differences in data cleaning,
- providing confidence to users of the final results of the analysis, that the researcher doing the work is competent, i.e., that cleaning was performed.

Ideally the operations performed on the original data, to transform it into what is considered a clean state, are collected together as a script for ease of replication.

Some cleaning activities are trivial and yet need to be performed to prevent the analysis being overwhelmed by what appears to be many special cases. For instance, an analysis<sup>59</sup> of different company's response to vulnerabilities reported in their products, started from raw data that sometimes contained slightly different ways of naming the same company. The company names data had to be cleaned to ensure that a single name was consistently used to denote each organization (see `data-check/patch-behav.R`).

The NASA Metrics Data Program (MDP) dataset contains fault data on 13 projects, and has been widely used by researchers (a literature survey for the period 2000 to 2010<sup>650</sup> found that 58 out of 208 fault prediction papers used it). This dataset contains many problems<sup>617</sup> that need to be sorted out, e.g., columns with all entries having the same value (suggesting a measurement or conversion error has occurred), duplicate rows, missing values (many occurred in rows calculated from other rows and involved a divide by zero), inconsistent values (e.g.,

---

<sup>ii</sup> 63% of the US population can be uniquely identified using only gender, ZIP code and full date of birth.<sup>588</sup>

number of function calls being greater than the number of operators) and nonsensical values (e.g., lines of code having fractional values).

Despite the non-trivial work needed to clean the MDP dataset, to remove spurious data, the authors of many of the papers using this dataset, have either not cleaned it, or only given a cursory summary of the cleaning operations<sup>178</sup> (e.g., “. . . removes duplicate tuples . . . along with tuples that have questionable values . . .”, does not specify what values were questionable). Consequently, even although the original dataset is publicly available it is difficult to compare the results published in different papers, because no information is available on what, if any, data cleaning operations were performed; so much of the data is in need of cleaning that any results based on an uncleaned version of this dataset must be treated with suspicion.

See [data-check/NASA\\_MDP-data\\_check.R](#) for examples of integrity checks performed on the MDP dataset.

It is possible that the values appearing in a sample are correct, but have been misclassified.

A survey of 682,000 unique Android devices in use during 2015, by OpenSignal,<sup>1193</sup> included the screen height and width reported by the device (upper plot in Figure 14.2). Many devices appear to have greater width than height, particularly those with smaller screens. Perhaps the device owners are viewing the OpenSignal website with their phones in landscape mode; the lower plot in Figure 14.2 switches the dimensions so that height has the larger value; switched values in red.

The fault repositories of open source projects are publicly available and the repositories of larger projects are a frequent source of data for fault analysis/prediction researchers.

A study by Herzig, Just and Zeller<sup>692</sup> manually classified over 7,000 issue reports extracted from the fault repositories of seven large Java projects. They found that on average 42.6% of reports had been misclassified, with 39% of files marked as defective not actually containing any reported fault (any fault prediction models built using the uncleaned data are likely to be misleading at best and possibly very wrong). An earlier analysis<sup>693</sup> had found that between 6 and 15% of bug fixing changes addressed more than one issue.

Possible reasons for the misclassification include: the status of an issue not being specified when the initial report is filed, resulting in the default setting of *Bug* being used; issue submitters having the opinion that a missing feature is a bug (request for enhancement was the most commonly reclassified status); and bug reporting systems only supporting a limited number of different issue statuses (forcing the submitter to use an inappropriate status).

The study also highlighted how much effort data cleaning consumes; the work was performed independently by two people and took a total of 725 hours (90 working days).

Sometimes the measured values from one or more subjects (e.g., people or programs) are remarkably different from the values measured for other subjects. It can be tempting to clean the data by removing the value for these subjects from the sample. A study by Müller and Höfer<sup>1116</sup> removed data on seven out of 18 subjects, because they considered the performance of these subjects was so poor that they constituted a threat to the validity of the experiment (whose purpose was to compare the performance of students and professional developers). This kind of activity might be classified as outlier removal or manipulating data to obtain a desired result, either way, documenting the cleaning activity makes it possible for readers of the analysis to decide.

## 14.2 Outliers

“An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism”.<sup>666</sup>

In some applications, observations that deviate from the general trend are the ones of interest,<sup>266</sup> e.g., intrusion detection and credit card fraud. This subsection covers the case where deviate observations are unwanted; some later subsections cover software engineering situations where deviate observations are themselves the subject of study.

Methods for handling outliers include:

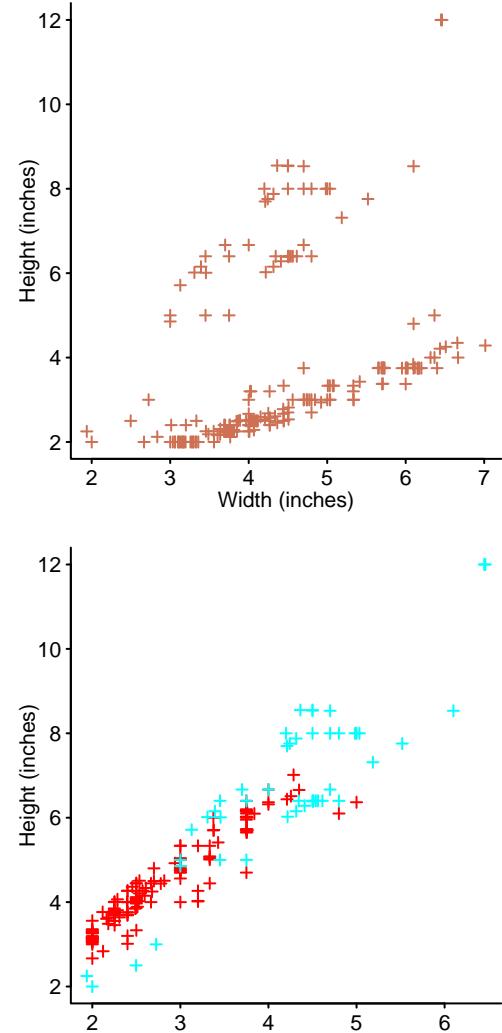


Figure 14.2: Screen height and width reported by 682,000 unique devices that downloaded an App from OpenSignal in 2015 (upper), reported measurements ordered so height always the larger value (lower). Data from OpenSignal.<sup>1193</sup> [code](#)

- using a statistical technique that does not assign too much weight to observations that deviate from the patterns followed by most other observations. Techniques capable of performing the desired statistical analysis are not always available, but when R functions implementing them are available they may be discussed in the appropriate section,
- detecting and excluding outliers from the subsequent analysis. Traditionally outliers have been manually selected and excluded from subsequent analysis. This approach can work well when the sample contains a small amount of data and the person doing the detection has sufficient domain knowledge. There are a variety of functions that automate the process of outlier selection and handling, some of these are discussed below (also see Section 11.2.6).

another definition of outlier detection is “ . . . the problem of finding patterns in data that do not conform to the expected normal behavior.”<sup>266</sup> This definition requires that an expected normal behavior be known, along with a method of comparing values for *outlyingness*.

Figure 14.3 shows a suspicious spike in the number of daily reported vulnerabilities recorded in the US National Vulnerability Database for 2003. What behavior could explain this pattern? Perhaps all vulnerabilities that had been reported, but not yet fully processed, were simply published, for the public to see, at the end of the year?

A study by Zheng, Mockus and Zhou<sup>1708</sup> investigated what they called problematic values, in the task completion time for Mozilla projects. A manual analysis found that for various reasons, some patches for reported faults were being committed in batches (so the commit date did not reflect the date the code for the patch was created). Enough information was available (i.e., there was data redundancy) to build a model that suggested values more likely to be correct (50% more accurate was claimed).

The date when an event occurred may appear unlikely, based on domain knowledge, e.g., staff rarely work at weekends. The following output shows a count of the number of features recorded as being Done, in a company using an Agile process,<sup>1</sup> for each day of the week. Monday is day 0 and the counts for Saturday/Sunday should be zero; the non-zero values suggest a 2-4% error rate, comparable with human error rates for low stress/non-critical work.

`code`

```
> table(Done_day %% 7)
  0   1   2   3   4   5   6 
 670 708 669 716 447 12  16
```

Should outliers be removed from the sample used for analysis?

While removing outliers may improve the quality of the model fitted to an equation, does it improve the quality of the fit of the model to reality?

Without understanding the processes that generated the data, there is no justification for removing any value.

The real issue with outliers is the impact they have on the final result. In a large sample, a few unusual values are unlikely to have any real impact data.

However, outliers are handled, any decision to exclude them from analysis needs to be documented.

## 14.3 Malformed file contents

A sign that data, or its organization in a file, is malformed in some way, is that the variable into which a file has been read does not have the expected contents (e.g., incorrect number of columns or the type of data in one or more columns is surprising, such as strings where a number should have occurred). The `str` function provides a quick and easy way of checking the types of columns in a data frame.

File formatting issues to watch out for include:

- functions for reading data in R (e.g., `read.csv` and `read.table`) often use the first few lines of the file being read as the format to use when reading the rest of the file, i.e., the number of columns contained in each row and the datatype of the values in each column.

If there are one or more rows that do not follow the format selected at the start of the file (e.g., different number of column delimiters; perhaps the result of non-delimited strings such as a missing pair of quote characters), then subsequent values may appear in the other columns, or be converted to a different type,

- termination delimiter missing from a string value; this can result in the contents of the following row being treated as part of the current row (because the newline is treated as part of the string),
- cut-and-pasting of data between media introducing conversion errors, e.g., the digit zero treated as the letter D or G during image to character conversion.

A variety of ad-hoc techniques are available for locating the cause of problems. For instance, the following code will convert all values that do not have the format of a number to NA, which are then easily located using base-library support for processing NAs, with their row index found using the `which` function:

```
which(is.na(as.number(as.character(data_frame$column_name))))
```

The `complete.cases` function returns a vector specifying which of the rows in its `data.frame` argument are complete, i.e., do not contain any NA values; the `na.omit` function returns a copy of its argument with any rows containing NA omitted.

## 14.4 Missing data

Missing data (for instance, a survey where the entry for a person's age is empty) is often the rule, rather than the exception, and books have been written on the subject. Missing data may be *disguised*, in that it appears as a reasonable value<sup>1226</sup> (e.g., zero when the range of possible legitimate values includes zero), or it may not be visible to the measurement process (e.g., intermittent check-ins to version control obscuring the detailed change history<sup>1141</sup>), or the input process provides a two item choice (e.g., male/female), with one item being the default and thus appearing as the missing value when no explicit choice is made.

The starting point for handling missing data is to normalise how it is denoted, to the representation used by R, i.e., NA (Not Available). Normalisation ensures that all missing values are treated consistently; special case handling of NA is built into R and many functions include options for handling NA.

A wide variety of different representations for missingness may be encountered (e.g., special values that cannot occur as legitimate data values such as 9999, "#N/A", "missing" or no value appearing between two commas in a comma separated list), and it is not uncommon for different columns within a dataset to use different representations (because they originate from different measurement sources).

The following code illustrates one method for changing a known representation of missing value to NA (the second form would be necessary if 9999 could appear as a legitimate value in a column other than `size`):

```
data[ data == 9999 ] = NA # set all elements having value 999 to NA

# set all elements of column size having value 999 to NA
data$size[ data$size == 9999 ] = NA
```

Once missing values have been explicitly identified it is possible to move on to deciding whether to ignore these cases or to replace NA with some numeric value. Some algorithms can handle missing values while others cannot; R functions vary in their ability to handle missing values. A few techniques for selecting the replacement value are discussed below.

The R base I/O functions, such as `read.csv`, have conventions for handling the case of zero characters appearing between the delimiters on each line of a file. The behavior depends on the type that has been assigned to values in a particular column. For columns assigned a numeric type, zero characters are treated as-if NA appeared between the delimiters, while for columns assigned a string type the zero character case is treated as the empty string rather than NA (i.e., treated the same as the string ""). These functions support a variety of options

for changing the default the handling of zero characters and the handling of leading/trailing white-space between delimiters.

As an example: reading a file containing the columns below left has the same effect as reading a file containing the columns below right:

X,Y_str,Z	X,Y_str,Z
1,"abc",2.2	1,"abc",2.2
2,,3.1	2,"",3.1
,NA,2	NA,NA,2

The `table` function counts occurrences of values, and by default does not include NA in the count; the `useNA` options has to be used to explicitly specify that NA be counted:

```
table(data$some_column, useNA="ifany") # limit the count to one column
```

This one column use can be expanded to cover every column in `data`. If the output is too voluminous, the number of columns processed can be reduced, or the call to `table` replaced by a call to `tabulate`, which provides more options to control behavior:

```
sapply(colnames(data), function(x) table(data[, x], useNA="ifany"))
```

While there may be documentation specifying how missing values are represented, such details may not be documented. An analysis of a dataset using the above code may show a suspiciously large number of values such as 9999 or -1 (for an attribute that can never be negative), a result that suggests further investigation is worthwhile.

#### 14.4.1 Handling missing values

When deciding what to do about missing values, it is important to try to understand why the values are missing. The following categories are commonly encountered in the analysis of missing data:

- Missing completely at random (MCAR): As the name suggests, the selection of missing values occurred completely at random. Statistically this is the most desirable kind of missingness, because it means there is no bias in the missing values,
- Missing at random (MAR): This sounds exactly like MCAR, but it is not completely random in the sense that the choice of which values are missing is influenced by other values in the sample. For instance, the level of seniority may correlate with the likelihood that survey questions about salary are answered,
- Missing not at random (MNAR): This missingness could be as random as MAR, with the one difference that the choice of missing values is influenced by values not in the sample. For instance, the name of the developer who originally wrote the code referenced in a fault report may be missing if that developer is a friend of the person reporting the fault, with friendship not being a recorded in the sample.

The following code can be used to get a rough estimate of the correlation between the rows of a `data.frame` that contain missing values (Figure 8.8 illustrates a method of visualizing this information):

```
x=is.na(some_data_frame)
# highlight rows having some, but not all, missing values
cor(subset(x, sd(x) > 0))
```

Many analysis techniques handle missing values by ignoring the rows or columns that contain them; if the sample contains many rows and a low percentage of missing values, this behavior may not be a problem. However, if the sample contains a large percentage of missing values, any analysis will either have to make do with a smaller number of measurements, be limited to using techniques that can gracefully adapt to missing data (i.e., don't ignore rows containing one or more missing values) or be forced to use estimated values for the missing data.

The ideal approach is to use an algorithm capable of handling samples that include missing data.

The process of estimating a value to use, where none is present in the sample, is known as *imputing*.

A quick and dirty method of imputing values, that can be effective, is to replace a missing value by the mean of the values in the corresponding column containing the missing value; alternatively, if the data is ordered in some way (e.g., dates), the last value appearing before the missing value might be used.

A more sophisticated approach to imputing values involves filling the missing value entries using other values present in the sample. The `Amelia`, `naniar` and `VIM` packages provide a variety of functions for visualizing datasets containing missing values and imputing values for these entries.

As Figure 11.66 shows, data that is missing over a complete range of values, rather than spread over the sample.

Data may be missing because the sample may not be large enough to be likely to contain instances of rarely occurring cases (which would be seen in a larger sample). Good-Turing smoothing<sup>537</sup> is a technique for adding non-zero counts to adjust for unseen items.

## 14.4.2 NA handling by library functions

R functions vary in their ability to handle `data.frames` containing `NA` and the behaviors exhibited include:

- behaving in unpredictable ways when `NA` is encountered,
- behaving in predictable ways, that perhaps is surprising to the unknowledgeable, e.g., the value of `NA == NA` is `NA`, as is `NA != NA`,

functions that operate on complete rows or columns have a variety of behaviors when they counter one or more `NA`s, including:

- supporting a parameter, often called `na.rm`, which can be used to select among various methods for handling any `NA` that occur,
- ignoring rows containing one or more `NA`, e.g., `glm` ignores these rows by default, but this behavior can be changed using the `na.action` option,
- making use of information present in rows containing one or more `NA`, e.g., the `rpart` function,

Some regression model building functions return information associated with individual data points, such as residuals. If the function removes rows containing any `NA` before building the regression model, the number of data rows included in the returned model may be less than originally passed in, unless rows containing `NA` is reinserted (e.g., by using the `naresid` function).

## 14.5 Restructuring data

When the data of interest is spread over several files, it may be necessary to read two or more files and merge their contents into a single `data.frame`.

If two datasets contain shared columns (i.e., column names, column ordering and information held are the same), the `rbind` function can be used to join rows together, returning a single `data.frame`; the `cbind` function performs the join operation for columns.

The `merge` function merges the contents of two `data.frames` based on one or more criteria, e.g., shared column names.

### 14.5.1 Reorganizing rows/columns

The organization of rows and columns in a `data.frame` may not be appropriate for that used by the library functions used to perform the analysis.

The values in a dataset are may be held in a wide format (i.e., a few rows and many columns), but a long format (i.e., many rows and a few columns) is required, or vice versa.

An example of wide format data is that used in Figure 2.5; the IQ test scores have the form:

```
test,gender,1,2,3,4,5,6,7,8,9
verbal,Boy,8455,14171,17596,29308,30490,27544,16037,9857,4635
verbal,Girl,5448,10570,15312,28591,32385,30830,18557,11443,5321
quantitative,Boy,3138,19634,18258,29037,23255,30376,16504,12565,5095
quantitative,Girl,2313,16905,19002,32707,26438,32413,15215,10007,3406
non-verbal,Boy,1390,18144,20713,29245,25720,27077,18095,11369,6077
non-verbal,Girl,1165,14370,18564,30488,29342,30458,18387,10450,5075
CAT3,Boy,2505,14505,19556,29917,29607,30327,17960,9392,2787
CAT3,Girl,1813,10927,17872,31059,32867,33269,18016,9041,2394
```

The `melt` function, in the `reshape2` package, transforms `data.frames` to a long format, such as the following (only the first 11 lines are shown):

	test	gender	stanine	count
1	verbal	Boy	X1	8455
2	verbal	Girl	X1	5448
3	quantitative	Boy	X1	3138
4	quantitative	Girl	X1	2313
5	non-verbal	Boy	X1	1390
6	non-verbal	Girl	X1	1165
7	CAT3	Boy	X1	2505
8	CAT3	Girl	X1	1813
9	verbal	Boy	X2	14171
10	verbal	Girl	X2	10570
11	quantitative	Boy	X2	19634

which was reorganized using the call (where `b_g_IQ` contains the data):

```
b_g=melt(b_g_IQ, id.vars=c("test", "gender"),
           variable.name="stanine", value.name="count")
```

It is also possible to convert from long to wide format.

## 14.6 Miscellaneous issues

### 14.6.1 Application specific cleaning

The analysis of some kinds of data has acquired established preprocessing procedures; the data is not wrong, but transforming it in some way improves the quality of the subsequent analysis. For instance, before analyzing text, common low interest words (such as “the” and “of”, known as *stop words*) are removed; also words may be stemmed (a process that removes suffixes with the intent of uncovering the root word, e.g., kicked and kicking both become kick).

### 14.6.2 Different name, same meaning

Typos in character based data may be detected because of constraints on what can appear in domain specific sequences (e.g., the spelling of words). More difficult to detect problems include different people using different terminology for the same concept, or the same terminology for different concepts.

The SPEC 2006 benchmark results often include a description of the characteristics of the memory used by the computer under test. For historical marketing reasons two scales are commonly used to specify memory performance; the DDR scale is based on peak bandwidth while the PC scale uses clock rate. The SPEC result descriptions are not consistent in their choice of scale and so before any analysis can be performed the values need to be converted to a single form. Also, for marketing reasons the numbers have been rounded to reduce the number of non-zero digits; an analysis interested in high accuracy would map all the *marketing* values to their actual values (see `benchmark/scripts/SPEC-memory.awk`).

An email address is sometimes the only unique identifying information available, e.g., the list of developers who have contributed to an open source project. The same person may have

used more than one email address over the period of their involvement in a project and it is necessary to detect which addresses belong to the same person. The `find.aliases` function in the `tm.plugin.mail` package attempts to detect which email addresses refer to the same person.

### 14.6.3 Multiple sources of signals

Sometimes a value appearing in a sample could have come from multiple sources, only one of which is of interest. An example of this is the question: when did hexadecimal literals first appear as such in print?

One way of answering this question is to analyze the word n-grams (and associated year of book publication) Google have made available from their English book scanning project.<sup>595</sup>

The regular expression `^ [0o0[xX] [0-9a-fA-F01]]` (ohh, Ohh and ell were treated as the corresponding digits) returned 89 thousand matches.

OCR mistakes have resulted in some words being treated as hexadecimal literals, e.g., Oxford was sometimes scanned as 0xf0fd. The character sequence oxo is surprisingly common, and looking at some of the contexts in which this sequence occurs suggests that the usage is mainly related to chemical formula (some uses are also likely to be references to a cooking product of this name).

Assuming that hexadecimal notation did not start appearing in books before electronic computers were invented, books prior to say 1945 (i.e., the end of World War II) can be ignored.

Your author also assumed that, if any hexadecimal literal appears in a book, at least one more such literal is likely to appear; applying this final filtering rule, the number of matches was reduced to 7,292; with 319 unique character sequences.

Figure 14.4 shows a comparison of the use of hexadecimal literals in C source with those extracted from Google books n-grams.

### 14.6.4 Duplicate data

Duplicate data can cause some analysis techniques to fail (e.g., regression modeling) or skew the calculated results.

Duplication data is easily generated: the collation of data from multiple sources can result in the same measurements appearing more than once, and there may be multiple measurements of the same event (e.g., logging of computer faults where a single root cause produces the same message at sporadic times after the fault is experienced<sup>1530</sup> and spatially or functionally adjacent units to generate messages;<sup>954</sup> see [data-check/Blue-Gene.log](#)).

The `duplicated` function returns information about rows that are exact duplicates. More subtle duplication may involve the values in a row/column differing by a constant factor from those in another row/column, e.g., one row contains temperature in Celsius while another uses Fahrenheit.

When the data is numeric, `close_duplicated` can be highlighted using pairwise correlation; see Section 10.5.4.

Some R functions handle duplicate row/columns gracefully (e.g., the `glm` function), while others give unpredictable results (e.g., the `solve` function which inverts a matrix), the behavior depends on the algorithm used and what if any consistency checks were added by the implementer of the code.

### 14.6.5 Default values

Sometimes a measurement process returns what is considered to be a reasonable value, if it cannot return the actual value. For instance, IP geolocation services are always able to associate a country with an IP address, but when they are unable to further refine the location within a country, they return a location near the center of the country; for the USA this is close to the town of Potwin in Kansas (population 449) which appears to experience orders of magnitude more Internet related events, for its population size, than other towns in the US.<sup>766</sup>

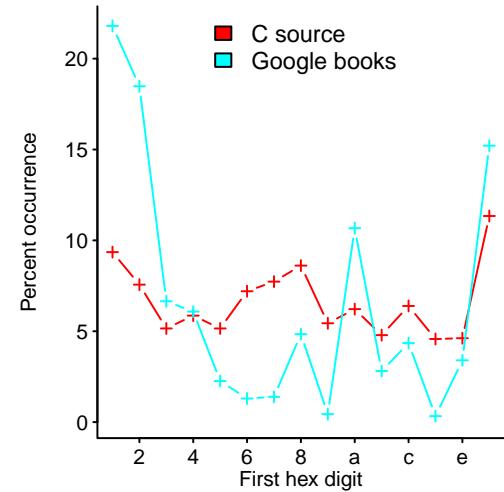


Figure 14.4: Percentage occurrence of the first digit of hexadecimal numbers in C source and estimated from Google book data. Data from Jones<sup>785</sup> and Michel et al.<sup>1075</sup> code

### 14.6.6 Resolution limit of measurements

Some kinds of measurement are inherently inexact, e.g., time. When working close to the resolution limit of the measuring process, false signals can be generated by the interaction between the measurement resolution and the processes generating measurement events.

A study by Feitelson<sup>480</sup> measured the runtime of processes, executed on a system, to an accuracy of two decimal digits. Initial analysis of the number of processes whose execution fell within a given time interval found a surprising behavior, there were many time intervals that did not contain any processes (upper plot in Figure 14.5). Further analysis found that the timer resolution was 1/64 second and the gaps were an artefact of the number of digits recorded, recording more digits (lower plot in Figure 14.5) resulted in fewer intervals containing no measurement points.

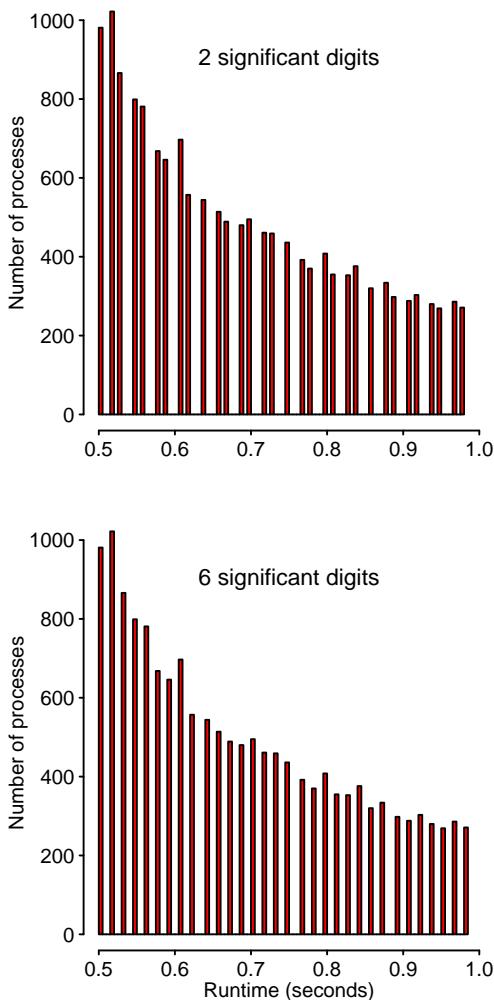


Figure 14.5: Number of processes executing for a given amount of time, with measurements expressed using two and six significant digits. Data from Feitelson.<sup>480</sup> [code](#)

## 14.7 Detecting fabricated data

A sample is not always derived from accurate measurements, the accuracy failures may be accidental or intentional, or might not involve any actual measurements (i.e., it has been fabricated).

Like all data analysis, detection of fabricated data is based on finding known patterns in the data (i.e., patterns that have previously been found to appear in known fabricated data). As always, the interpretation of why the data contains these patterns is the responsibility of the audience of the results; it is always worth repeating that domain knowledge is key.

One pattern of behavior that has been observed to occur in real world data, with some regularity, is that the first digit of numeric values follows Benford's law to a reasonable degree of approximation (while a figure of 30% of all datasets has been quoted, the actual figure is likely to be much smaller<sup>1400</sup>). The failure of data to follow Benford's law has been used to detect accounting and election<sup>1355</sup> fraud, identification of fake survey interviews<sup>1393</sup> and scientific research.<sup>403</sup>

While references to Benford's law usually involve the first digit of numeric values, there is a form that applies to the second and perhaps other significant digits.<sup>1158</sup> There has also been work<sup>125</sup> suggesting that the digit at the opposite end of numeric literals, the least significant digit, sometimes has a uniform distribution.

Benford's law specifies that the probability of the first digit having value  $d$  is given by:  $P(d) = \log_{10}(1 + \frac{1}{d})$

Figure 7.35 shows counts of the first digit of numeric literals in C source code.

If a set of independent and identically distributed random variables are sorted, the distribution of digits of the differences between adjacent sorted values is close to Benford's law.<sup>1086</sup> A test based on this fact can detect rounded data, data generated by linear regression and data generated by using the inverse function of a known distribution.<sup>1158</sup>

The BenfordTests package contains a variety of function for evaluating the conformity of a dataset to Benford's law.

When generating fabricated data, it is sometimes necessary to produce a random sequence of items. People hold incorrect beliefs<sup>173</sup> about the properties of random sequences and when asked to generate them produce sequences that contain predictable patterns (i.e., they are not random).

One study<sup>1395</sup> was able to build a model that predicted repeated patterns in an individual's randomly selected sequence, with around 25% success rate, but when the model built for one person's behavior was used to make predictions about another persons the success rate dropped to around 18%.

Detecting divergence from, or agreement with, these patterns of behavior depends on the authors of the data being unfamiliar with the expected patterns of behavior, or being lazy (i.e., being unwilling to spend the time making sure that the data they generate has the expected characteristics; the creators of the fictitious accounts publicly published by Madoff's companies, before his fraud was uncovered, made the effort to ensure that they followed Benford's law<sup>1404</sup>).

An excess of round numbers has been used to suggest that data has been fabricated.<sup>871</sup>

# Chapter 15

## Overview of R

This chapter gives a brief overview of R for developers who are fluent in at least one other computer language. The discussion pays attention to language features that are very different from languages the reader is likely to be familiar with; the focus is on a few language features that can be used to solve most problems.

The R language is defined by its one implementation; available from the R core team.<sup>1299</sup> A language definition,<sup>1300</sup> written in English prose, is gradually being written.

R programs tend to be very short, compared to programs in languages such as C++ and Java; 100 lines is a long R program. It is assumed that most readers will be casual users of R, whose programs generally follow the pattern:

```
d=read_data()
clean_d=clean_and_format(d)
d_result=applicable_statistical_routine(clean_d)
display_results(d_result)
```

If your problem cannot be solved using this algorithm, then the most efficient solution may be for you, dear reader, to use the languages and tools you are already familiar with, to preprocess the data so that it can be analysed and processed using R.

R is a domain specific language, whose designers have done an excellent job of creating a tool suited to the tasks frequently performed when analysing the kinds of datasets encountered in statistical analysis. Yes, R is Turing complete, so any algorithm that can be implemented in other programming languages can be implemented in R, but it has been designed to do certain things very well, with no regard to making it suitable for general programming tasks.

As a language the syntax and semantics of R is a lot smaller than many other languages. However, it has a very large base library, containing over 1,000 functions. Most of the investment needed to become a proficient user of R has to be made in learning to how to combine these functions to solve the problem at hand. There are over 10,000+ add-on packages available from the CRAN (Comprehensive R Archive Network).

Help on a specific identifier, if any is available, can be obtained using the ? (question mark) unary operator, followed by the identifier. The ?? unary operator, followed by the identifier, returns a list of names associated with that identifier for which a help page is available.

The call `library(help=circular)`, lists the functions and objects provided by the package named in the argument.

### 15.1 Your first R program

Much like Python, Perl and many other interpreted implementations, R can be run in an interactive mode, where code can be typed and immediately executed (with "Hello world" producing the obvious output).

Your first R program ought to read some data and plot it, not just print "Hello World". The following program reads a file containing a single column of values and plots them, to produce Figure 15.1:

```
the_data=read.csv("hello_world.csv")
```

```
plot(the_data)
```

The `read.csv` function is included in the library that comes bundled with the base system (functions not included in this library have to be loaded using the `library` function, before they can be referenced; the package containing them may also need to be installed via the `install.packages` function) and has a variety of optional arguments (arguments can be omitted if the function definition includes default values for the corresponding parameter). Perhaps the most commonly used optional arguments are `sep` (the character used to separate, or delimit, values on a line, defaults to comma) and `header` (whether the contents of the first line should be treated as column names, default `TRUE`).

The value returned by `read.csv` has class `data.frame`, which might be thought of as a C struct type (it contains data only, there are no member functions as such).

The `plot` function attempts to produce a reasonable looking graphic of whatever data is passed, which for character data is a histogram of the number of occurrences. Users of R are not expected to be interested in manipulating low level details and some effort is needed to get at the numeric values of characters.

There are a wide variety of options to change the appearance of `plot` output; these can be applied on each call to `plot`, or globally for every call (using the `par` function).

All objects in the current environment can be saved to a file using the `save` function, and a previously saved environment can be restored using the `load` function. When quitting an R session, the user is given the option of saving the current environment to a file named `.RData`; if a file of this name exists, when R is started, its contents are automatically loaded.

## 15.2 Language overview

R is a language and an environment. Like Perl, it is defined by how its single implementation behaves (i.e., the software maintained by the R project<sup>[1299](#)</sup>).

R was designed, in the mid-1990s, to be largely compatible with S (a language, which like C, started life in the mid-1970s at Bell Labs). When S was created, Fortran was the dominant engineering language and the Fortran way of doing things had a strong impact on early design decisions (i.e., R does not have a C view of the world; for instance, it uses a row/column, rather than column/row ordering).

The designers of R have called it a functional language, and it does support a way of doing things that is most strongly associated with functional program languages (including making life cumbersome for developers wanting to assign to global variables).

The language also contains constructs that are said to make it an object oriented language, and it certainly contains some features found in object oriented languages. OO constructs were first added in the third iteration of the S language and are more of an addition to the functional flavor of the language than a complete make-over. The primary OO feature usage is function overloading, when accessing functions from library packages.

Lateral thinking is often required to code a problem in R, using knowledge of functions contained in the base system, e.g., calling order to map a vector of strings to a unique vector of numbers.

Some data analysts write non-trivial programs in R, which means they have to deal with the testing and debugging issues experienced by users of in other languages.

Base library support for debugging includes support for single stepping through a function, via the `debug` function, and setting breakpoints via the `setBreakpoint` function; package support includes: RUnit package for unit testing, and the covr package for measuring code coverage.

### 15.2.1 Differences between R and widely used languages

The following list describes language behaviors that are different from that encountered in other commonly used languages (Fortran developers will not consider some of these to be differences):

- there are no scalars, e.g., 2 is a vector containing one element and is equivalent to writing `c(2)`. Most unary and binary operators operate on all elements of a vector (among other things).

Many operations that involve iterating over scalar values in other languages, e.g., adding two arrays, can be performed without explicit iteration in R, e.g., `c(1, 2) + c(3, 4)` has the value `c(4, 6)`,

- arrays start at one, not zero,
- matrices and data frames are indexed in row-column order (C-like languages use column-row order),
- case is significant in identifiers, e.g., `some_data` and `Some_data` are considered to refer to different objects,
- the period (dot, full stop, i.e., `.`) is a character than can occur in identifiers (e.g., `a.name`), it is not a separate token having the role of an operator,
- some language constructs, implemented via specific language syntax in other languages, are implemented as function calls in R, e.g., the functionality of `return` and `switch` is provided by function calls,
- assignment to a variable in an outer scope, from within a function, is specified using the `<->` operator. The other assignment operators (e.g., `<-`, `->` and `=`) always assign to a local variable (creating one, if a variable of the given name does not already exist, in local scope),
- vectors/arrays/data.frames can be sliced to return a subset of the original,
- explicit support for NA (Not Available). This value denotes a number that may exist, but whose value is unknown. Operations involving NA, return NA, when the result value is not known because the value of NA is unknown, but will return a value when the result is independent of the value of NA, e.g., `NA || TRUE`.
- type conversion behavior may be driven by semantics rather than the underlying representation, e.g., `as.numeric("1") == 1` and `as.numeric("a")` returns NA.

The following R language features are found in commonly used languages:

- objects and functions come into existence, during program execution, when they are assigned a value, appear as a function parameter, or in more obscure ways (there is no mechanism for declaring any kind of identifier),
- the type of an object is the type of the value last assigned to it,
- decimal and hexadecimal literals have type numeric (literals starting with zero are not treated as octal literals; any leading zero is ignored) even if they look like integers, because they do not contain a decimal point. Some input functions, e.g., `read.csv`, consider a column to have integer type, if all its values can be represented as an integer,
- what most other languages consider to be a statement (i.e., something that does not return a value) R treats as an expression (e.g., if/for statements return a value).

## 15.2.2 Objects

Operations in R are performed on objects, sometimes known as variables. Objects are characterized by their names and their contents; with the contents in turn being characterized by attributes specifying the kind of data contained in the object.

The R type system has evolved over time, and includes the terms *mode* (a higher level view of the value representation, at least sometimes, than *typeof*, e.g., `integer` and `double` have mode `numeric`), *storage.mode* (a concept going back to the S language) and *typeof* (the underlying representation used by the C implementation of the language).

The `mode`, `storage.mode` and `typeof` functions return a string containing the respective information, e.g., `numeric`, `integer` or `function`.

The length of an object is the number of elements it contains (e.g., a two-dimensional array containing  $i$  rows and  $j$  columns, contains  $i \times j$  elements). The `length` function returns the number of elements in its argument.

The assignment operator creates an object, with the object name being the left operand and its value and type being that of the right operand (left/right is reversed when the  $\rightarrow$  assignment is used).

## 15.3 Operations on vectors

### 15.3.1 Creating a vector/array/matrix

An R vector can be thought of as a one-dimensional array. Vectors are indexed starting at 1 (not zero) and it is possible to add additional elements to a vector, but not remove an existing element.

```
x = 2                      # new vector containing one value
x = c(2, 4, 6, 8, 10)      # new vector containing five values
# new vector containing the contents of x and two values
x = c(x, 12, 14)
y = vector(length=5)        # new vector created by function call
y = 3:8                     # same as c(3, 4, 5, 6, 7, 8)
z = seq(from=3, to=13, by=3) # create a sequence of values
# All elements converted to a common type
z = c(1, 2, "3")           # String has the greater conversion precedence
```

Multidimensional arrays can be created using the `array` function, with the common case of 2-dimensional arrays supported by a specific function, i.e., the `matrix` function.

```
> # create 3-dimensional array of 2 by 4 by 6, initialized to 0
> a3=array(0, c(2, 4, 6))
> matrix(c(1, 2, 3, 4, 5, 6), ncol=2) # default, populate in column order
[1,] [2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> # specify the number of rows and populate by row order
> matrix(c(1, 2, 3, 4, 5, 6), nrow=2, byrow=TRUE)
[1,] [2] [3]
[1,]    1    2    3
[2,]    4    5    6
> x = matrix(nrow=2, ncol=4) # create a new matrix
> y = c(1, 2, 3)
> z = as.matrix(y) # convert a vector to a matrix
> str(y)
num [1:3] 1 2 3
> str(z)
num [1:3, 1] 1 2 3
```

### 15.3.2 Indexing

One or more elements of a vector/array/matrix can be accessed using indexing. Accesses to elements that do not exist return NA. Negative index values specify elements that are excluded from the returned value.

The zeroth element returns an empty vector.

```
> x = 10:19
> x[2]
[1] 11
> x[-1]                      # exclude element 1
[1] 11 12 13 14 15 16 17 18 19
> x[12]                        # there is no 12'th element
[1] NA
```

```
> x[12]=100          # there is now
> x
[1] 10 11 12 13 14 15 16 17 18 19 NA 100
```

Multiple elements can be returned by an indexing operation:

```
> x = 20:29
> x[c(2,5)]          # elements 2 and 5
[1] 21 24
> y = x[x > 25]      # all elements greater than 25
> y
[1] 26 27 28 29
> # The expression x > 25 returns a vector of boolean values
> i = x > 25
> i
[1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE
> # an element of x is returned if the corresponding index is TRUE
> x[i]
[1] 26 27 28 29
```

Matrix indexing differs from vector indexing in that out-of-bounds accesses generate an error.

```
> x = matrix(c(1, 2, 3, 4, 5, 6), ncol=2)
> x[2, 1]
[1] 2
> # x[2, 3] need to be able to handle out-of-bounds subscripts in Sweave...
> x[, 1]
[1] 1 2 3
> x[1, ]
[1] 1 4
> x[3, ]=c(0, 9)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    0    9
> x=cbind(x, c(10, 11, 12))  # add a new column
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
> x=rbind(x, c(5, 10, 20))  # add a new row
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
[4,]    5   10   20
```

### 15.3.3 Lists

The difference between a list and a vector is that different elements in a list can have different modes (types) and existing elements can be removed.

```
> x = list(name="Bill", age=25, developer=TRUE)
> x
$name
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE
> x$name
[1] "Bill"
```

```

> x[[2]]
[1] 25
> x = list("Bill", 25, TRUE)
> x
[[1]]
[1] "Bill"

[[2]]
[1] 25

[[3]]
[1] TRUE
> y = unlist(x) # convert x to a vector, all elements are converted to strings
> y
[1] "Bill" "25"    "TRUE"
> x = list(name="Bill", age=25, developer=TRUE)
> x$sex="M"  # add a new element
> x
$name
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE

$sex
[1] "M"
> x$age = NULL # remove an existing element
> x
$name
[1] "Bill"

$developer
[1] TRUE

$sex
[1] "M"

```

The `[[ ]]` operator returns a value, while `[ ]` returns a sublist (which has mode list).

### 15.3.4 Data frames

From the perspective of a programmer coming from another language, it may seem appropriate to think of a data frame as behaving like a matrix, and in some cases it can be treated in this way (e.g., when all columns have the same appropriate type functions expecting a matrix argument may work). However, a better analogy is to think of it as an indexable structure type (where different members can have different types).

The `read.csv` function reads a file containing columns, of potentially different types, and returns a data frame.

When indexing a data frame like a matrix, elements are accessed in row-column order (not the column-row order used in C-like languages). The following code selects all rows for which the `num` column is greater than 2.

```

> x = data.frame(num=c(1, 2, 3, 4), name=c("a", "b", "c", "d"))
> x
  num name
1   1    a
2   2    b
3   3    c
4   4    d
> # Middle two values of the column named num
> x$num[2:3]
[1] 2 3

```

```

> # Have to remember that rows are indexed first and also specify x twice
> x[x$num > 2, ]
  num name
3   3    c
4   4    d
> # Using the subset function removes the possibility of making common typo mistakes
> # No need to remember row/column order and only specify x once
> subset(x, num > 2)
  num name
3   3    c
4   4    d

```

If one or more columns contain character mode values (i.e., strings), `read.csv` will create, by default, a factor rather than a vector. The argument: `as.is=TRUE` causes strings to be represented as such.

Where ever possible, the code written for this book uses the `subset` function, rather than relying on correctly indexing a data frame.

### 15.3.5 Symbolic forms

An R expression can have a value which is its symbolic form.

```

exp = expression(x/(y+z))
eval(expr) # evaluate expression using the current values of x, y and z

```

Uses of expression values include: specifying which vectors in a table to plot in a graph, and including equations in graphs, for instance:

```
text(x, y, expression(p == over(1, 1+e^(alpha*x+beta))))
```

will cause the following equation to be displayed at the point (x, y).

$$p = \frac{1}{1 + e^{\alpha x + \beta}}$$

The `D` function takes an expression as its first argument, and based on the second argument, returns its derivative:

```

> D(expression(x/(y + z)^2), "z")
-(x * (2 * (y + z)) / ((y + z)^2)^2

```

### 15.3.6 Factors and levels

When manipulating non-numeric values (e.g., names) statisticians sometimes find it convenient to map these values to integer values and manipulate them as integers. In programming terminology, a variable used to represent one or more of these integer values could be said to have a *factor* type, with the actual numeric values known as *levels* (a parallel can be drawn with the enumeration types found in C++ and C, except these assign names to integer values).

```

> factor(c("win", "win", "lose", "win", "lose", "lose"))
[1] win win lose win lose lose
Levels: lose win

```

Some operations implicitly convert a sequence of values to a factor. For instance, `read.csv` will, by default, convert any column of string values to a factor; this conversion is a simplistic form of hashing, and (when a megabyte was considered a lot of memory) was once driven by the rationale of saving storage. These days the R implementation uses more sophisticated hashing, and we live with the consequences of historical baggage.

Operations of objects holding values represented as factors sometimes have surprising effects, for those unaware of how things used to be.

## 15.4 Operators

Operators in R have the same precedence rules as Fortran, which in some cases differ from the C precedence rules (which most commonly used languages now mimic). An example of this difference is: `!x ==y` which is equivalent to `!(x ==y)` in R, but equivalent to `(!x) ==y` in C-like languages (if x and y have type boolean, there is no effective difference, but expressions such as: `!1 ==2` produce a different result).

A list of operators and their precedence can be obtained by typing `?Syntax`, at the R command line.

Within an expression operand evaluation is left to right, except assignment which evaluates the right operand and then the left.

In most cases, all elements of a vector are operated on by operators:

```
> c(5, 6) + 1
[1] 6 7
> c(1, 2) + c(3, 4)
[1] 4 6
> c(7, 8, 9, 10) + c(11, 12)
[1] 18 20 20 22
> c(0, 1) < c(1, 0)
[1] TRUE FALSE
```

in the last two examples *recycling* occurs, that is the elements of the shorter vector are reused until all the elements of the longer vector have been operated on.

The `&&` and `||` operators differ from `&` and `|` in that they operate on just the first element of their operands, returning a vector containing one element, e.g., `c(0,1) && c(1,1)` returns the vector `FALSE`.

The base system includes a set of `bitw???` functions, that perform bitwise operations on their integer arguments; there is the `bitops` package.

Operators	Description
<code>:: :::</code>	access variables (right operand) in a name space (left operand)
<code>\$ @</code>	component / slot extraction (member selection has lower precedence than subscripting in C-influenced languages)
<code>[ [</code>	array and list indexing
<code>^ **</code>	exponentiation (associates right to left)
<code>- +</code>	unary minus and plus
<code>:</code>	sequence operator
<code>%any%</code>	special operators ( <code>%%</code> and <code>/%%</code> has the same precedence as <code>*</code> and <code>/</code> in C-influenced languages)
<code>* /</code>	multiply and divide
<code>+ -</code>	(binary) add and subtract
<code>&lt; &gt; &lt;= &gt;= == !=</code>	relational and equality (non-associative; equality has lower precedence in C-influenced languages)
<code>!</code>	negation (greater precedence than any binary operator in C-influenced languages)
<code>&amp; &amp;&amp;</code>	and of all elements and the first element
<code>  </code>	or of all elements and the first element
<code>~</code>	as in formulae
<code>-&gt; -&gt;&gt;</code>	local and global rightwards assignment
<code>=</code>	assignment (associates right to left)
<code>&lt;- &lt;&lt;-</code>	local and global assignment (associates right to left)
<code>?</code>	help (unary and binary)

Table 15.1: Operators listed in precedence order.

The character used for exclusive-or in C-influenced languages, `^`, is used for exponentiation in R; the `xor` function performs an exclusive-or of its operands. Like Fortran, R also supports the use of `**` to denote exponentiation.

The `[` and `[[` operators differ by more than being array and list indexing. The result of the index `x[1]` has the same type as `x` (i.e., the operation preserves the type), while the result of `x[[1]]` is a simplified version of the type of `x` (if simplification is possible).<sup>i</sup>

```
> x = c(a = 1, b = 2)
> x[1]
a
1
> x[[1]]
[1] 1
> x = list(a = 1, b = 2)
> str(x[1])
List of 1
 $ a: num 1
> str(x[[1]])
num 1
> x = matrix(1:4, nrow = 2)
> x[1, ]
[1] 1 3
> x[1, , drop = FALSE]
 [,1] [,2]
[1,] 1 3
> # x[[1, ]] is not allowed
>
> df = data.frame(a = 1:2, b = 1:2)
> str(df[1])
'data.frame': 2 obs. of 1 variable:
 $ a: int 1 2
> str(df[[1]])
int [1:2] 1 2
> str(df[, "a", drop = FALSE])
'data.frame': 2 obs. of 1 variable:
 $ a: int 1 2
> str(df[, "a"])
int [1:2] 1 2
```

### 15.4.1 Testing for equality

In addition to the equality operators, the base system includes two equality related functions, `identical` and `all.equal`.

```
> x = 1:5 ; y = 1:5
> x == y # Return the result of equality test for each corresponding element
[1] TRUE TRUE TRUE TRUE TRUE
> identical(x, y) # Return a single value denoting exact equality
[1] TRUE
> 1L == 1 # 1L is stored internally as an integer, 1 is stored as a double
[1] TRUE
> identical(1L, 1) # identical requires the stored type be the same
[1] FALSE
> 0.9 == (1.1 - 0.2) # could be affected by lack of precision
[1] FALSE
> all.equal(0.9, 1.1 - 0.2) # do a fuzzy compare
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance=0) # find out much how fuzz there is
[1] "Mean relative difference: 1.233581e-16"
```

The default tolerance used by the `all.equal` function is `.Machine$double.eps^0.5`.

Comparisons against `NA` always returns `NA`. The `is.na` function can be used to check for this quantity; the `anyNA` function returns `TRUE`, if its argument contains at least one `NA`.

### 15.4.2 Assignment

Four of the ways of assigning a value to a variable in R are:

---

<sup>i</sup> Out-of-bounds handling is also different, but I'm sure readers' don't do that sort of thing.

```
x <- 3 # Operator used by people who follow the herd
x <<- 3 # Assigns to the x at global scope

3 -> x # Rarely encountered outside descriptions of the language

x = 3 # Supported since R version 1.4
```

Many R books and articles use the two characters `<-`.<sup>ii</sup> Developers are used to seeing the `=` token, and with nothing other than conformity to existing R usage to recommend the alternative, the assignment token that developers are already very familiar with, is used in this book.

There is one context where `=` does not behave like normal assignment. R supports the use of parameter names in arguments to function calls, to explicitly specify that a named parameter is to be assigned a given value. In the context of a function argument list, the left operand of `=` is treated as the name of a parameter and the right operand as the value to be assigned. An error is flagged, if the function definition does not have a parameter having the specified name.

```
func = function (a, b, c) a + b * c

func(2, 3, 9)
func(c=9, b=3, a=2)

func(d=3, 4, 5) # no parameter named d, an error is raised

# use <- if the intent is to assign to d and pass this value as an argument
func(d<-3, 4, 5)
```

## 15.5 The R type (mode) system

R supports values having the following basic types (R also has the concept of *mode*, which is based on semantics rather than underlying representation, e.g., the `mode` function returns `numeric` where `typeof` returns either `integer` or `double`):

- `NULL`:
- `raw`: essentially uninterpreted byte values,
- `logical`: holds one of the values: `TRUE`, `FALSE`, `T` or `F`. The conversion `as.logical(any_n_on_zero_value)` returns `TRUE`,
- `character`: what many other languages call a string type,
- `integer`: the only integer type, contains 32 bits (NA is represented using the most negative value, so this value is not available as an integer; trying to generate this or any other value outside of the representable value of a 32-bit integer will result in a value having a double type),
- `double`: the only floating-point type, contains 64 bits,
- `complex`: contains a real and imaginary double type,

An object may be reported to have one of these basic types, but it may actually be a vector or array of this type.

More complicated types may be created, such as lists, data frames, etc.

---

<sup>ii</sup> The developers of the S language used terminals that had a single key for this sequence.

### 15.5.1 Converting the type (mode) of a value

It is often possible to convert the mode (type) of a value by calling the `as.some_mode` function, where `some_mode` is the name of a mode, e.g., `integer`. If a conversion fails, `NA` is returned.

#### Conversion precedence

```
NULL < raw < logical < integer < real < complex < character < list < expression
```

## 15.6 Statements

R contains the usual language constructs that look like statements, but they can behave like expressions:

- **function**: defines a function, whose value has to be assigned to an object:  
`f=function(p1, p2) {return(p1+p2)}`
- blocks of code are bracketed using the punctuation pair: `{` and `}`,
- `;` (semicolon) is required to delimit multiple expressions on the same line, but is otherwise optional,
- **if**: which takes an optional **else** arm (there is no **then** keyword, but there is an **ifthenelse** function),
- **for**: which has the form `for (i in x)`, where `x` is a vector (such as `1:10`),
- **while** and **repeat** loops are available,
- loops may be terminated using the **break** keyword or the `break` function, and may be continued at the next iteration using the **next** keyword or `next` function,
- **return** is a function: `return(1+return(1))` returns the value 1,
- **switch** is a function.

## 15.7 Defining a function

```
> g=1 # a global variable
> f = function(p1, p2) # define a function and assign it to f
+ {
+ l=g # Value access, check lexical and dynamic scope for g
+ g=2 # Assignment: only check local scope, if no variable exists, create one
+
+ m=h # h is dynamically in scope
+
+ return(return(1)+1) # return is a function call
+ }
> h=2 # another global variable
> f(1, 2)
[1] 1
> g
[1] 1
> h=3 # At global scope, so must be global variable
```

Argument evaluation is lazy, that is, they are evaluated the first time their value is required.

The `...` token (three dots) specifies that a variable number of unknown arguments may be passed.

```
unk_args=function(...)
{
a=list(...) # Convert any arguments passed to a list of values
# Access the list of values in a
}
```

## 15.8 Commonly used functions

Technically every operation is a function call (so '+'(1, 2) and 1+2 are equivalent), but not all function calls have equivalent operator tokens.

```
> x = 1:10
> if (any(x > 7)) print("At least one value greater than 7")
[1] "At least one value greater than 7"
> if (all(x > 0)) print("All values greater than zero")
[1] "All values greater than zero"
> rep(1:2, 3)
[1] 1 2 1 2 1 2
```

- `head/tail` mimics the behavior of the unix `head/tail` programs,
- `length` returns the number of elements in its vector argument,
- `nrow/ncol` return the number of rows/columns in the data frame argument (NROW/NCOL gracefully handle vector arguments),
- `order` returns a vector containing an index in to the argument in the order needed to sort the argument values,
- `str` lists the columns in a variable, along with their type and the first few values in each row; it provides a quick way of verifying that columns have the expected type.
- `which` returns a vector of values containing the index of the argument values that are true,
- `methods`: list functions overloaded on the argument name
- `installed.packages`: list all installed packages
- `ls`: lists variables that exist in the current environment,
- `system.time, proc.time`: cpu time used and the real and cpu time of the currently running R process.

## 15.9 Input/Output

Functions are available for reading data having a variety of formats (e.g., comma separated values), from all the common data sources (e.g., files, databases, web pages). In some cases the contents of a compressed file will be automatically uncompressed before reading. In the case of files, all the data contained in the file is often read, and returned as a single object.

Many functions try to automatically deduce the datatype of the data read, e.g., whether it is integer, real, character sequence, etc. Sometimes the datatype selected is not correct, and work has to be done to ensure the data is treated as having the desired type; the `read.csv` function bases its decision on the type of each column, by analysing the first 6, or so, lines of the file.

Some functions in the base system, e.g., `read.csv`, convert columns containing string values to factors, by default; the original intent was, presumably, to reduce the storage needed to hold the data. A column of factors, as a type, does not always behave the same as a column of strings and this default conversion behavior is often a liability. Using the argument `as.is=TRUE` prevents values being converted to factors (it is used in all of this book's example code).

```
data=read.csv("measurements.csv.gz", as.is=TRUE) # file will be uncompressed
data=read.csv("measurements.csv", sep="|", as.is=TRUE) # change separator
data=read.csv("https://github.com/Derek-Jones/ESEUR-code-data/blob/master/benchmark/MST")
```

The first line of the input file is assumed to denote the name of each column, specifying `header=FALSE` switches off this default behavior.

All characters on an input line after, and including, the comment character, `#`, are ignored (various options interact with this behavior, including the `comment.char` option which can be used to change the character used).

The `foreign` package supports the reading (and some writing) of data stored in some of the binary representations used by other applications (e.g., `read.spss`).

If data is not already in a form that can be easily processed by R, it may be simpler to convert it using a language or tool that you are already familiar with, rather than using R.

The R environment includes a simple spreadsheet like editor for manual data entry and modifying existing data.

```
scores = edit(scores) # invoke built-in spreadsheet like editor
```

There are corresponding `write` functions for many of the `read` functions, e.g., `write.csv`.

The `print` function performs relatively simple formatted output (the `format` function can be used to create more sophisticated formatting, that can then be output); the `cat` function performs relatively little formatting, but is more flexible, and in particular does not terminate its output with a newline; the `sink` function can be used to specify an alternative location to write console output.

### 15.9.1 Graphical output

There are probably more functions supporting graphical output, in R, than textual output. Perhaps the most commonly used graphical output function is `plot`. This function often does a good job of producing a reasonable graphical representation of the data. Overloaded versions of this function are often provided by packages, to plot data having a particular class created by the package.

By default, graphical output is sent to the console device; this behavior can be overridden to produce a file having a particular format, e.g., `pdf`, `jpeg`, `png` and `pictex`. The list of supported output devices varies across the operating systems on which R runs.

The behavior of the `plot` function can be influenced by previous calls to the `par` function, which set configurable options.

Various packages providing graphical output are available, with the `ggplot` package probably being the most commonly used by frequent R users.

## 15.10 Non-statistical uses of R

While the target of R's domain specialised functionality is statistical data analysis, there are other application domains where this functionality could be useful (but may not warrant effort needed to learn R).

A variety of functions designed for manipulating the rows and columns of delimited data files are available; see [Rlang/Top500.R](#).

A technique for spotting whether a file contains compressed data (e.g., a virus hidden in a script by compressing it to look like a jumble of numbers) is to plot the fraction of distinct values appearing in successive, fixed size, blocks; see Figure 15.2. Compressed data is likely to contain an approximately uniform distribution of byte values (compression is achieved by reducing apparent information content), your mileage may vary between compression methods.

The following code reads a pdf file, applies a sliding window to the data and then plots the fraction of distinct values in each window (at a given offset).

```
window_width=256 # if less than 256, divisor has to change in plot call
```

```
plot_unique=function(filename)
{
```

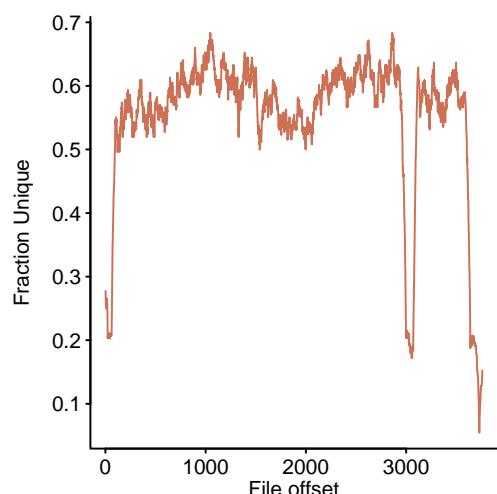


Figure 15.2: The unique bytes per window (256 bytes wide) of a pdf file. [code](#)

```
t=readBin(filename, what="raw", n=1e7)

# Sliding the window over every point is too much overhead
cnt_points=seq(1, length(t)-window_width, 5)

u=sapply(cnt_points, function(X) length(unique(t[X:(X+window_width)])))
plot(u/256, type="l", xlab="Offset", ylab="Fraction Unique", las=1)

return(u)
}

dummy=plot_unique("http://www.coding-guidelines.com/R_code/requirements.tgz")
```

## 15.11 Very large datasets

While most existing software engineering datasets tend to be small, exceptions may occur from time to time. A variety of techniques are available for handling large datasets, including:

- the `bigmemory` package provides software defined memory management (i.e., swapping data between memory and main storage). The `bigtabulate` package, along with other `big???` packages contain functions that perform commonly used operations on this data.
- the `data.table` package extends `data.frames` to support up to 100G of storage,

#NoEstimates#NoEstimates, 116 Time  
and Materials ContractsTime and Materials Contracts 120  
Structured Distributed Software Development: A Case StudyS-  
tructured Distributed Software Development: A Case Study 124  
Educator BeliefsEducator Beliefs 179



# Bibliography

1. 7digital, ltd??? website, July 2012. <http://www.7digital.com>. 128, 129, 222, 308, 309, 360
2. J. T. Abbott, J. L. Austerweil, and T. L. Griffiths. Random walks on semantic networks can resemble optimal foraging. *Psychological Review*, 122(3):558–569, July 2015. 30
3. T. Abdel-Hamid and S. E. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice-Hall, Inc, 1991. 115
4. D. Aboody and B. Lev. The value relevance of intangibles: The case of software capitalization. *Journal of Accounting Research*, 36:161–191, 1998. 75
5. ACAA Technical Agent. Ada conformity assessment test suite (ACATS). website, Jan. 2018. <http://www.ada-auth.org/acats.html>. 160
6. E. N. Adams. Optimizing preventive service of software products. *IBM Journal of Research and Development*, 28(1):2–14, Jan. 1984. 147, 148
7. J. Adams. *Risk and Freedom: The record of road safety regulation*. Transport Publishing Projects, 1985. 48
8. B. Adelson. Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9(4):422–433, July 1981. 30, 31
9. J. Agar. *The Government Machine A Revolutionary History of the Computer*. The MIT Press, 2003. 81
10. R. Agarwal and M. Gort. The evolution of markets and entry, exit and survival of firms. *The Review of Economics and Statistics*, 78(3):489–498, Aug. 1996. 89
11. P. J. Ågerfalk. Insufficient theoretical contribution: a conclusive rationale for rejection? *European Journal of Information Systems*, 23(6):593–599, Nov. 2014. 6
12. A. Aghayev and P. Desnoyers. Skylight-A window on shingled disk operation. In *13th USENIX Conference on File and Storage Technologies*, FAST’15, pages 135–149, Feb. 2015. 350
13. N. Agrawal. *Representative, reproducible, and practical benchmarking of file and storage systems*. PhD thesis, University of Wisconsin-Madison, 2009. 341
14. N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Storage*, 5(4):125–138, Dec. 2009. 204
15. N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *ACM Transactions on Storage*, 3(3):31–45, Oct. 2007. 225
16. N. Ahmad. Measuring investment in software. OECD Science, Technology and Industry Working Papers 2003/06, OECD, May 2003. 74
17. J. J. Ahonen and P. Savolainen. Software engineering projects may fail before they are started: Post-mortem analysis of five cancelled projects. *Journal of Systems and Software*, 83(11):2175–2187, Nov. 2010. 110
18. F. Akdemir and F. A. Kirmani. Synergy: A synthetic study on teams. Thesis (m.s.), Umeå School of Business, July-Sept. 2008. 73
19. G. A. Akerlof. The market for "Lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, Aug. 1970. 70
20. K. Akita, S. Itagaki, Y. Masawa, M. Nonaka, T. Hatani, K. Hattori, S. Morisaki, Y. Yanagida, T. Takaya, T. Furuyama, and O. Takashi. *Software Development Data White paper 2012-2013*. SEC BOOKS, 2012. 102, 109
21. H. A. A. Al-Mutawa. On the classification of cyclic dependencies in Java programs. Thesis (m.s.), Massey University, New Zealand, 2013. 188
22. H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, July 2013. 272, 273, 278
23. N. Ali, Z. Sharifi, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study on requirements traceability using eye-tracking. In *28th IEEE International Conference on Software Maintenance*, ICSM’12, pages 191–200, Sept. 2012. 25
24. R. C. Allen. The British industrial revolution in global perspective: How commerce created the industrial revolution and modern economic growth. *unpublished???, ???(???)????, Apr. 2007*. 81
25. L. Allodi. Economic factors of vulnerability trade and exploitation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS’17, pages 1483–1499, Oct.-Nov. 2017. 143
26. L. Allodi and F. Massacci. A preliminary analysis of vulnerability scores for attacks in wild: The EKITS and SYM datasets. In *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*, DABGERS’12, pages 17–24, Oct. 2012. 143
27. D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye. Do software developers understand open source licenses? In *25th IEEE International Conference on Program Comprehension*, ICPC 2017, pages 1–11, May 2017. 62
28. M. G. Almiron, E. S. Almeida, and M. N. Miranda. The reliability of statistical functions in four software packages freely used in numerical computation. *Brazilian Journal of Probability and Statistics*, 23(2):107–119, 2009. 12
29. M. G. Almiron, B. Lopes, A. L. C. Oliveira, A. C. Medeiros, and A. C. Frery. On the numerical accuracy of spreadsheets. *Journal of Statistics*, 34(4):1–29, Apr. 2010. 12
30. L. E. Alteneder. The learning curve in solving a jig-saw puzzle: A teaching device. *Journal of Educational Psychology*, 26(3):231–232, Mar. 1935. 32
31. E. M. Altmann. *Episodic Memory for External Information*. PhD thesis, Carnegie Mellon University, Aug. 1996. 26
32. E. M. Altmann. Functional decay of memory for tasks. *Psychological Research*, 66(4):287–297, 2002. 22
33. E. M. Altmann, J. G. Trafton, and D. Z. Hambrick. Effects of interruption length on procedural errors. *Journal of Experimental Psychology: Applied*, 23(2):216–229, June 2017. 29
34. H. Aman, S. Amasaki, T. Yokogawa, and M. Kawahara. A survival analysis of source files modified by new developers. In *International Conference on Product-Focused Software Process Improvement*, PROFES 2017, pages 80–88, Nov.-Dec. 2017. 153
35. Amazon, Inc. Amazon ec2 service level agreement. <https://aws.amazon.com/ec2/sla/>, June 2013. 354
36. S. Ambler. IT project success survey results. <http://www.ambysoft.com/surveys/>, 2017. 110
37. J. M. Amiri and V. V. K. Padmanabhu. A comprehensive evaluation of conversion approaches for different function points. Thesis (m.s.), Blekinge Institute of Technology, Sweden, Sept. 2011. 273, 274
38. L. An, O. Mlouki, F. Khomh, and G. Antoniol. Stack overflow: A code laundering platform? In *eprint arXiv:cs.SE/1703.03897*, Mar. 2017. 118
39. L. V. B. An T. Oskarsson, G. H. McClelland, and R. Hastie. What's next? Judging sequences of binary events. *Psychological Bulletin*, 135(2):262–285, 2009. 17, 46
40. B. C. D. Anda, D. I. K. Sjøberg, and A. Mockus. Variability and reproducibility in software engineering: A study of four companies that developed the same system. *IEEE Transactions on Software Engineering*, 35(3):407–429, May 2009. 110, 118
41. C. Anderson, J. A. D. Hildreth, and L. Howland. Is the desire for status a fundamental human motive? A review of the empirical literature. *Psychological Bulletin*, 141(3):574–601, May 2015. 67
42. J. R. Anderson. *Learning and Memory: An Integrated Approach*. John Wiley & Sons, Inc, second edition, 2000. 35
43. J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, Apr. 2007. 17

44. J. R. Anderson and R. Milson. Human memory: An adaptive perspective. *Psychological Review*, 96(4):703–719, 1989. 31
45. M. L. Anderson. Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–313, Apr. 2010. 15
46. D. Andriesse, X. Chen, V. van der Veen, A. Slowinska, and H. Bos. An in-depth analysis of disassembly on full-scale x86/x64 binaries. In *Proceedings of the 25th USENIX Security Symposium*, pages 583–600, Aug. 2016. 160
47. J. Annett. Subjective rating scales: science or art? *Ergonomics*, 45(12):966–987, 2002. 356
48. A. Ansar. ‘AppStore secrets’ (What we’ve learned from 30,000,000 downloads). Presentation, pinch media, 2009. 99
49. F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, Feb. 1973. 273, 275
50. K. Aoki and M. W. Feldman. Evolution of learning strategies in temporally and spatially variable environments: A review of theory. *Theoretical Population Biology*, 91:3–19, Feb. 2014. 67
51. J. Aranda. Anchoring and adjustment in software estimation. Thesis (m.s.), Graduate Department of Computer Science, University of Toronto, 2005. 113
52. L. Argote, C. A. Insko, N. Yovetich, and A. A. Romero. Group learning curves: The effects of turnover and task complexity on group performance. *Journal of Applied Social Psychology*, 25(6):512–529, Mar. 1995. 69
53. H. R. Arkes, R. M. Dawes, and C. Christensen. Factors influencing the use of a decision rule in a probabilistic task. *Organizational Behavior and Human Decision Processes*, 37:93–110, 1986. 50
54. P. Armer. SHARE – A eulogy to cooperative effort. Technical Report P-969, The RAND Corporation, Oct. 1956. 62, 101
55. J. S. Armstrong. The seer-sucker theory: The value of experts in forecasting. *Technology Review*, pages 16–24, June-July 1980. 34
56. V. Arnaoudova, L. M. Eshkevari, M. Di Penta, R. Oliveto, G. Antoniol, and Y.-G. Guéhéneuc. REPENT: Analyzing the nature of identifier renamings. *IEEE Transactions on Software Engineering*, 40(5):502–532, May 2014. 177
57. J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2010. 184
58. T. B. Arnold and J. W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, 3(2):34–39, Dec. 2011. 218
59. A. Arora, R. Krishnan, R. Telang, and Y. Yang. An empirical analysis of software vendors’ patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, Mar. 2010. 317, 319, 358
60. W. B. Arthur. Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal*, 99(394):116–131, Mar. 1989. 91
61. W. B. Arthur. *Increasing Returns and Path Dependency in the Economy*. The University of Michigan Press, 1994. 89, 91
62. S. E. Asch. Studies of independence and conformity: A minority of one against a unanimous majority. *Psychological Monographs: General and Applied*, 70(9):1–70, 1956. 49
63. T. A. Åstebro, S. A. Jeffrey, and G. K. Adomdza. Inventor perseverance after being told to quit: The role of cognitive biases. *Journal of Behavioral Decision Making*, 20(3):253–272, Apr. 2007. 50
64. S. Atkinson and G. Benefield. Software development: Why the traditional contract model is not fit for purpose. In *46th Hawaii International Conference on System Sciences*, HICSS, pages 4842–4851, Jan. 2013. 120
65. V. Atlidakis, J. Andrus, R. Geambasu, D. Mitropoulos, and J. Nieh. POSIX abstractions in modern operating systems: The old, the new, and the missing. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys’16, page 19, Apr. 2016. 105
66. Audit Scotland. i6: a review. Report, Audit Scotland, Mar. 2017. 110
67. Auerbach. Auerbach guide to time sharing. Computer technology report, Auerbach Publishers Inc., Jan. 1973. 94
68. N. R. Augustine. *Augustine’s Laws*. American Institute of Aeronautics and Astronautics, Inc, sixth edition, 1997. 145
69. R. Auler and E. Borin. A LLVM just-in-time compilation cost analysis. Technical Report IC-13-13, Instituto de Computação Universidade Estadual de Campinas, May 2013. 165, 166
70. P. C. Austin. A tutorial on multilevel survival analysis: Methods, models and applications. *International Statistical Review*, 85(2):185–203, Aug. 2017. 319
71. R. D. Austin. The effects of time pressure on quality in software development: An agency model. *Information Systems Research*, 12(2):195–207, June 2001. 71
72. J. L. Autran, D. Munteanu, P. Roche, and G. Gasiot. Real-time soft-error rate measurements: A review. *Microelectronics Reliability*, 54(8):1455–1476, Aug. 2014. 155
73. J.-L. Autran, S. Semikh, D. Munteanu, S. Serre, G. Gasiot, and P. Roche. Soft-error rate of advanced SRAM memories: Modeling and monte carlo simulation. In M. Andriychuk, editor, *Numerical Simulation – From Theory to Industry*, chapter 15, pages 309–336. InTech, Sept. 2012. 155
74. E. Avidan. The significance of method parameters and local variables as beacons for comprehension: An empirical study. Thesis (m.s.), The Hebrew University of Jerusalem, Nov. 2016. 177
75. P. Azoulay, C. Fons-Rosen, and J. S. G. Zivin. Does science advance one funeral at a time? Working Paper No. 21788, National Bureau of Economic Research, USA, Dec. 2015. 8
76. R. H. Baayen, P. Milin, and M. Ramscar. Frequency in lexical processing. *Aphasiology*, 30(11):1174–1220, Mar. 2016. 175
77. V. Babka. *Improving Accuracy of Software Performance Models on Multicore Platforms with Shared Caches*. PhD thesis, Faculty of Mathematics and Physics, Charles University in Prague, Oct. 2012. 352
78. V. Babka and P. Tůma. Investigating cache parameters of x86 family processors. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, pages 77–96, Jan. 2009. 352, 353
79. D. Baccarini, G. Salm, and P. E. D. Love. Management of risks in information technology projects. *Industrial Management & Data Systems*, 104(4):286–295, 2004. 121
80. A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE’13, pages 712–721, May 2013. 159
81. A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2010, pages 97–106, Nov. 2010. 141
82. J. Backus. The history of FORTRAN I, II, and III. *SIGPLAN Notices*, 13(8):165–180, 1978. 102
83. J. Backus. Programming in America in the 1950s— some personal impressions. In N. Metropolis, J. Howlett, and G.-C. Rota, editors, *A History of Computing in the Twentieth Century*, pages 125–135. Academic Press, Feb. 1981. 101, 102
84. J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, H. Herrick, R. A. Hughes, L. B. Mitchell, R. A. Nelson, R. Nutt, D. Sayre, P. B. Sheridan, H. Stern, and I. Ziller. *The FORTRAN Automatic Coding System for the IBM 704 EDPM: Programmer’s Reference Manual*. International Business Machines Corporation, 590 Madison Avenue, New York 22, N.Y., Oct. 1956. 102
85. A. Bacon, S. Handley, and S. Newstead. Individual differences in strategies for syllogistic reasoning. *Thinking & Reasoning*, 9(2):133–168, 2003. 40
86. A. Baddeley. Working memory. In A. Baddeley, M. W. Eysenck, and M. Anderson, editors, *Memory*, chapter 3, pages 41–69. Psychology Press, Feb. 2009. 27
87. A. Baddeley. Working memory: Theories, models, and controversies. *Annual Review of Psychology*, 63:1–29, Sept. 2012. 27
88. A. D. Baddeley, N. Thomson, and M. Buchanan. Word length and the structure of short-term memory. *Journal of Verbal Learning and Verbal Behavior*, 14(6):575–589, Dec. 1975. 27, 341
89. M. Bagherzadeh, N. Kahani, C.-P. Bezemer, A. E. Hassan, J. Dingel, and J. R. Cordy. Analyzing a decade of Linux system calls. *Empirical Software Engineering*, 23(3):1519–1551, June 2018. 105
90. J. N. Bailenson, M. S. Shum, S. Atran, D. L. Medin, and J. D. Coley. A bird’s eye view: biological categorization and reasoning within and across cultures. *Cognition*, 84:1–53, 2002. 38

91. D. H. Bailey. Misleading performance reporting in the supercomputer field. Technical Report RNR-92-005, Numerical Aerodynamic Simulation Division, NASA Ames Research Center, Dec. 1992. 346
92. S. Baily, R. Gilbertson, and E. Straub. Modular multimode radar (CMMR) software acquisition study. Technical Report 2302-01-1-2291, ARINC Research Corporation, Mar. 1981. 97
93. E. Bainomugisha, A. L. Carreton, T. van Cutsem, S. Mostinckx, and W. de Meuter. A survey on reactive programming. *ACM Computing Surveys*, 45(4):52, Aug. 2013. 164
94. P. Bajari, S. Tadelis, and S. Houghton. Bidding for incomplete contracts: An empirical analysis of adaptation costs. *American Economic Review*, 101(4):1288–1319, Oct. 2011. 119
95. A. H. Baker, D. M. Hammerling, M. N. Levy, H. Xu, J. M. Dennis, B. E. Eaton, J. Edwards, C. Hannay, S. A. Mickelson, R. B. Neale, D. Nychka, J. Shollenberger, J. Tribbia, M. Vertenstein, and D. Williamson. A new ensemble-based consistency test for the Community Earth System Model (pyCECT v1.0). *Geoscientific Model Development*, 8:2829–2840, Sept. 2015. 139
96. F. T. Baker. Chief programmer team management of production programming. *IBM Systems Journal*, 11(1):56–73, 1972. 132
97. M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, May 2002. 255
98. B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems (HotPower’12)*, Oct. 2012. 299
99. N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *International Conference on Ubiquitous Computing, UbiComp 2007*, pages 217–237, Sept. 2007. 84
100. P. Banyard and N. Hunt. Something missing? *The Psychologist*, 13(2):68–71, 2000. 17
101. L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li. Who will leave the company?: A large-scale industry study of developer turnover by mining monthly work report. In *14th IEEE/ACM International Conference on Mining Software Repositories, MSR’17*, pages 170–181, May 2017. 62
102. J. H. Barkow, L. Cosmides, and J. Tooby. *The Adapted Mind: Evolutionary Psychology and the Generation of Culture*. Oxford University Press, 1992. 16
103. W. P. Barnett. *The Red Queen among Organizations: How competitiveness evolves*. Princeton University Press, 2008. 90
104. A. Baronchelli, V. Loreto, and A. Puglisi. Individual biases, cultural evolution, and the statistical nature of language universals: The case of colour naming systems. *PLoS ONE*, 10(5):e0125019, May 2015. 181
105. D. R. Barrett. *World Christian Encyclopedia: A Comparative Survey of Churches and Religions in the Modern World AD 1900-2000*. Oxford University Press, 1982. 86
106. L. Barrett, R. Dunbar, and J. Lycett. *Human Evolutionary Psychology*. Palgrave Macmillan, 2002. 16
107. L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Report, Morgan & Claypool, 2009. 84
108. V. R. Basili and J. Beane. Can the Parr curve help with manpower distribution and resource estimation problems? *The Journal of Systems and Software*, 2(1):59–69, Feb. 1981. 115
109. V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz. The empirical investigation of perspective-based reading. In *Proceedings of the Twentieth Annual Software Engineering Workshop*, pages 21–69, Dec. 1995. 6, 339
110. V. R. Basili, N. M. Panlilio-Yap, C. L. Ramsey, C. Shih, and E. E. Katz. A quantitative analysis of software developed in Ada. Technical Report TR-1403, Department of Computer Science, University of Maryland, May 1984. 45
111. V. R. Basili and A. J. Turner. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4):390–396, Dec. 1975. 123
112. F. M. Bass. A new product growth model for consumer durables. *Management Science*, 15(5):215–227, Jan. 1969. 78
114. H. A. Bastiaanse. *Very, Many, Small, Penguins: Vaguely Related Topics*. PhD thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, Mar. 2014. 45
115. B. Baudry, S. Allier, and M. Monperrus. Tailored source code transformations to synthesize computationally diverse program variants. In *eprint arXiv:cs.SE/1401.7635v1*, Jan. 2014. 152, 179
116. F. L. Bauer and H. Wössner. The "Plankalkül" of Konrad Zuse: a forerunner of today's programming languages. *Communications of the ACM*, 15(7):678–685, July 1972. 102
117. A. Baumann. Hardware is the new software. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS’17*, pages 132–137, May 2016. 107
118. R. F. Baumeister. *Is There Anything Good About Men?* Oxford University Press, 2010. 17
119. R. T. Baust. *Computer Characteristics Quarterly: Volume 7, Number 4-Volume 8, Number 1*. adams associates, 1968. 84
120. G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. The evolution of project inter-dependencies in a software ecosystem: the case of Apache. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance, ICSM’13*, pages 280–289, Sept. 2013. 92
121. G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. How the Apache community upgrades dependencies: An evolutionary study? *Empirical Software Engineering*, 20(5):1275–1317, Oct. 2015. 92, 93
122. O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The influence of non-technical factors on code review. In *20th Working Conference on Reverse Engineering, WCRE 2013*, pages 122–131, Oct. 2013. 134
123. B. L. Bayus, S. Jain, and A. G. Rao. Truth or consequences: An analysis of vaporware and new product announcements. *Journal of Marketing Research*, 38(1):3–13, Feb. 2001. 70
124. A. A. Beaujean. *Latent Variable Modeling Using R*. Routledge, 2014. 355
125. B. Beber and A. Scacco. What the numbers say: A digit-based test for election fraud. *Political Analysis*, 20(2):211–234, Apr. 2012. 366
126. C. Becker, F. Fagerholm, R. Mohanani, and A. Chatzigeorgiou. Temporal discounting in technical debt: How do software practitioners discount the future? In *eprint arXiv:cs.SE/1901.07024*, Jan. 2019. 51
127. G. S. Becker. Investment in human capital: A theoretical analysis. *Journal of Political Economy*, 70(5):9–49, Oct. 1962. 65
128. R. A. Becker and W. S. Cleveland. *Trellis Graphics User’s Manual*. AT&T Bell Laboratories, Murray Hill, Dec. 1995. 203
129. J. Beckhusen. Occupations in information technology. American Community Survey Report ACS-35, U.S. Census Bureau, Aug. 2016. 98
130. A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. Technical Report MSR-TR-2013-111, Microsoft Research, Oct. 2013. 5, 356
131. M. Bekoff, C. Allen, and G. M. Burghardt. *The Cognitive Animal: Empirical and Theoretical Perspectives on Animal Cognition*. MIT Press, 2002. 16
132. C. G. Bell. Fundamentals of time shared computers. *Computer Design*, 7(2):44–59, Feb. 1968. 1
133. C. G. Bell. The mini and micro industries. *Computer*, 17(10):14–30, Oct. 1984. 83
134. G. Bell. Bell’s law for the birth and death of computer classes: A theory of the computer’s evolution. MSR-TR 2007-146, Microsoft Research, Silicon Valley, Nov. 2007. 83
135. G. Bell. Supercomputers: The amazing race (A history of supercomputing, 1960-2020). Technical Report MSR-TR-2015-2, Microsoft Research, Microsoft Corporation, Nov. 2014. 83
136. V. A. Bell and P. N. Johnson-Laird. A model theory of modal reasoning. *Cognitive Science*, 22(1):25–51, 1998. 40
137. M. Beller, A. Zaidman, A. Karpov, and R. A. Zwaan. The last line effect explained. *Empirical Software Engineering*, 22(3):1508–1536, June 2017. 118, 152
138. R. W. Bemer. A view of the history of COBOL. *Honeywell Computer Journal*, 5(3):130–135, Nov. 1959. 102

139. O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3), Sept. 2013. [57](#)
140. G. Beniamini, S. Gingichashvili, A. K. Orbach, and D. G. Feitelson. Meaningful identifier names: The case of single-letter variables. In *25th IEEE International Conference on Program Comprehension*, ICPC 2017, pages 45–54, May 2017. [95](#)
141. J. R. Beniger. *The Control Revolution: Technological and Economic Origins of the Information Society*. Harvard University Press, 1986. [3](#)
142. Y. Benkler. Coase's Penguin, or, Linux and the nature of the firm. *The Yale Law Journal*, 112(3), Dec. 2002. [55](#), [80](#)
143. A. Benson, D. Li, and K. Shue. Promotions and the Peter principle. SSRN Working Paper n. 3047193, Feb. 2018. [97](#)
144. R. A. Bentley, C. P. Lipo, H. A. Herzog, and M. W. Hahn. Regular rates of popular culture change reflect random copying. *Evolution and Human Behavior*, 28(3):151–158, May 2007. [68](#)
145. F. C. Y. Benureau and N. P. Rougier. Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. In *eprint arXiv:cs.GL/1708.08205*, Aug. 2017. [103](#)
146. T. Berger, S. She, K. Czarnecki, and A. Wąsowski. Feature-to-code mapping in two large product lines. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 498–499. Springer Berlin Heidelberg, 2010. [219](#)
147. T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. Variability modeling in the systems software domain. Technical Report GSDLAB-TR 2012-07-06, Generative Software Development Laboratory, University of Waterloo, July 2012. [183](#)
148. E. Berghout, M. Nijland, and K. Grant. Seven ways to get your favoured IT project accepted – politics in IT evaluation. *The Electronic Journal of Information Systems Evaluation*, 8(1):31–40, 2005. [110](#)
149. B. Berlin and P. Kay. *Basic Color Terms: Their Universality and Evolution*. Berkeley: University of California Press, 1969. [181](#)
150. D. Bermbach and E. Wittern. Benchmarking web API quality. In *2016 International Conference on Web Engineering*, ICWE'16, pages 188–206, June 2016. [157](#)
151. A. Bernardo and I. Welch. On the evolution of overconfidence and entrepreneurs. *Journal of Economics & Management Strategy*, 10(3):301–330, 2001. [67](#)
152. K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, July 2006. [345](#)
153. D. M. Berry, K. Daudjee, J. Dong, I. Fainchtein, M. A. Nelson, T. Nelson, and L. Ou. User's manual as a requirements specification: Case studies. *Requirements Engineering*, 9(1):67–82, Feb. 2004. [122](#)
154. D. M. Berry, E. Kamsties, and M. M. Krieger. From contract details to software specification: Linguistic sources of ambiguity. Nov. 2003. [151](#)
155. L. M. A. Bettencourt, A. Cintrón-Arias, D. I. Kaiser, and C. Castillo-Chávez. The power of a good idea: Quantitative modeling of the spread of ideas from epidemiological models. *Physica A*, 364:513–536, May 2006. [68](#)
156. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory-JCDT'99: 7th International Conference*, pages 217–235. Springer-Verlag, Jan. 1999. [329](#)
157. D. Biber, S. Johansson, G. Leech, S. Conrad, and E. Finegan. *Longman Grammar of Spoken and Written English*. Pearson Education, 1999. [40](#), [151](#), [178](#)
158. B. Biddle, A. White, and S. Woods. How many standards in a laptop? (and other empirical questions). SSRN Working Paper n. 1619440, Arizona State University (ASU) - College of Law, Sept. 2010. [73](#)
159. S. Biddle. Like everyone else, Twitter hides from U.S. taxes in Ireland. website: accessed: 5-Feb-17, Oct. 2013. <http://valleywag.gawker.com/like-everyone-else-twitter-hides-from-u-s-taxes-in-ir-1447085830>. [75](#)
160. B. Biegel, F. Beck, W. Hornig, and S. Diehl. The order of things: How developers sort fields and methods. In *28th IEEE International Conference on Software Maintenance*, ICSM'12, pages 88–97, Sept. 2012. [186](#), [187](#), [335](#)
161. S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *Journal of Political Economy*, 100(5):992–1026, Oct. 1992. [49](#)
162. P. Bilton, P. Dodimead, E. Livingstone, I. Rayner, G. Turner, M. Wynniatt, and S. Howes. Managing the risks of legacy ICT to public service delivery. HC 539 SESSION 2013-14, National Audit Office, UK, Sept. 2013. [85](#), [134](#)
163. W. L. Bircher. *Predictive Power Management for Multi-Core Processors*. PhD thesis, The University of Texas at Austin, Dec. 2010. [348](#), [350](#)
164. C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced? Bias in bug-fix datasets. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, FSE 2009, pages 121–130, Aug. 2009. [141](#)
165. S. Bird. Software knows best: A case for hardware transparency and measurability. Thesis (m.s.), Department of Electrical Engineering and Computer Science, University of California at Berkeley, May 2010. [341](#)
166. P. G. Bishop and R. E. Bloomfield. Worst case reliability prediction based on a prior estimate of residual defects. In *Proceedings 13th International Symposium on Software Reliability Engineering*, ISSRE'02, pages 295–303, Nov. 2002. [148](#)
167. T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *37th Annual International Computer Software & Applications Conference*, COMPSAC 2013, pages 303–312, July 2013. [131](#), [200](#)
168. Bitsavers' pdf document archive. Document archive, July 2019. <http://bitsavers.trailing-edge.com/pdf/>. [104](#)
169. N. M. Blachman. A survey of automatic digital computers. Survey 111293, Office of Naval Research, Washington, D.C., 1953. [102](#), [345](#)
170. S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khan, K. S. McKinley, R. Bentzur, A. Divan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis (extended version). Technical Report TR-CS-06-01, Department of Computer Science, Australian National University, Aug. 2006. [248](#)
171. A.-R. Blais and E. U. Weber. A domain-specific risk-taking (DOSPERT) scale for adult populations. *Judgment and Decision Making*, 1(1):33–47, Apr. 2006. [48](#)
172. M. S. Blaubergs and M. D. S. Braine. Short-term memory limitations on decoding self-embedded sentences. *Journal of Experimental Psychology*, 102(4):745–748, 1974. [28](#)
173. D. S. Blinder and D. M. Oppenheimer. Beliefs about what types of mechanisms produce random sequences. *Journal of Behavioral Decision Making*, 21(4):414–427, Oct. 2008. [366](#)
174. N. Bloom, T. Kretschmer, and J. van Reenen. Are family-friendly workplace practices a valuable firm resource? *Strategic Management Journal*, 32(4):343–367, Apr. 2011. [98](#)
175. B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Inc, 1981. [273](#), [274](#)
176. B. W. Boehm, T. E. Gray, and T. Seewaldt. Prototyping vs. specifying: A multi-project experiment. Technical Report 82-500, Center for Software Engineering, University of Southern California, 1982. [114](#)
177. B. W. Boehm and P. N. Papaccio. A value-chain analysis of software productivity. Technical Report USC-CSE-86-500, Center for Systems and Software Engineering, University of Southern California, 1986. [76](#)
178. G. D. Boetticher. Improving credibility of machine learner models in software engineering. In D. Zhang and J. J. P. Tsai, editors, *Advances in Machine Learning Applications in Software Engineering*, chapter 3, pages 52–73. Idea Group Publishing, Oct. 2006. [359](#)
179. J. G. Bolten, R. S. Leonard, M. V. Arena, O. Younossi, and J. M. Sollinger. Sources of weapon system cost growth: Analysis of 35 major defense acquisition programs. Monograph series, RAND Corporation, 2008. [113](#)

180. C. F. Bond, Jr. and L. J. Titus. Social facilitation: A meta-analysis of 241 studies. *Psychological Bulletin*, 94(2):265–292, Sept. 1983. [72](#)
181. J. Bonvoisin, R. Mies, J.-F. Boujut, and R. Stark. What is the "source" of open source hardware? *Journal of open hardware*, 1(1):5, Sept. 2017. [64](#)
182. R. Bornat, S. Dehnadi, and S. ??? Mental models, consistency and programming aptitude. In *Tenth Australasian Computing Education Conference*, ACE'08, pages 53–61, Jan. 2008. [164](#)
183. L. Boroditsky. Metaphoric structuring: understanding time through spatial metaphors. *Cognition*, 75:1–28, 2000. [96](#)
184. A. Börsch-Supan and M. Weiss. Productivity and age: Evidence from work teams at the assembly line. Technical Report 148-2007, Manheim Research Institute for the Economics of Aging, 2007. [53](#)
185. L. Bossavit. *The Leprechauns of Software Engineering: How folklore turns into fact and what to do about it*. Leanpub, 2016. [74](#)
186. N. Bostrom and A. Sandberg. The wisdom of nature: An evolutionary heuristic for human enhancement. In J. Savulescu and N. Bostrom, editors, *Human Enhancement*, chapter 18, pages 375–416. Oxford University Press, Jan. 2011. [16](#)
187. A. Botchkarev. Estimating the accuracy of the return on investment (ROI) performance evaluations. *Interdisciplinary Journal of Information, Knowledge, and Management*, 10:217–233, 2015. [56](#)
188. J. Bourn. New IT systems for Magistrates' courts: the Libra project. Report by the Comptroller and Auditor General HC 327 Session 2002-2003, National Audit Office, UK, Jan. 2003. [119](#)
189. E. M. Bowden and M. Jung-Beeman. Normative data for 144 compound remote associate problems. *Behavior Research Methods, Instruments, & Computers*, 35(4):634–639, Dec. 2003. [72](#)
190. G. H. Bower, J. B. Black, and T. J. Turner. Scripts in memory for text. *Cognitive Psychology*, 11(2):177–220, Apr. 1979. [170](#)
191. J. S. Bowers and C. J. Davis. Bayesian just-so stories in psychology and neuroscience. *Psychological Bulletin*, 138(3):389–414, 2012. [18](#), [232](#)
192. R. Boyd and P. J. Richerson. Why does culture increase human adaptability. *Ethology and Sociobiology*, 16(2):125–143, Mar. 1995. [67](#)
193. R. Boyd and P. J. Richerson. Why culture is common, but cultural evolution is rare. *Proceedings of the British Academy*, 88:77–93, Apr. 1996. [67](#)
194. M. G. Bradac, D. E. Perry, and L. G. Votta. Prototyping a process monitoring experiment. *IEEE Transactions on Software Engineering*, 20(10):774–784, 1994. [124](#), [125](#)
195. T. F. Brady, T. Konkle, G. A. Alvarez, and A. Oliva. Visual long-term memory has a massive storage capacity for object details. *PNAS*, 105(38):14325–14329, Sept. 2008. [52](#)
196. D. Braha and Y. Bar-Yam. The statistical mechanics of complex product development: Empirical and analytical results. *Management Science*, 53(7):1127–1145, July 2007. [164](#), [165](#)
197. D. W. Braithwaite and R. L. Goldstone. Flexibility in data interpretation: effects of representational format. *frontiers in Psychology*, 4(980):1–16, Dec. 2013. [202](#)
198. N. R. Bramley. *Constructing the world: Active causal learning in cognition*. PhD thesis, University College London, Feb. 2017. [43](#)
199. M. C. Branco, Y. Xiong, K. Czarnecki, J. Küster, and H. Völzer. An empirical study on consistency management of business and IT process models. Technical Report GSMLAB-TR 2012-03-02, Generative Software Development Laboratory, University of Waterloo, Mar. 2012. [92](#)
200. S. Brand. *How buildings Learn: What happens after they're built*. Viking, 1994. [122](#)
201. J. D. Bransford and J. J. Franks. The abstraction of linguistic ideas. *Cognitive Psychology*, 2(4):331–350, Oct. 1971. [169](#), [170](#)
202. J. D. Bransford and M. K. Johnson. Contextual prerequisites for understanding: Some investigations of comprehension and recall. *Journal of Verbal Learning and Verbal Behavior*, 11(6):717–726, Dec. 1972. [168](#), [169](#)
203. G. Branwen. Laws of tech: Commoditize your complement. Gwern blog, Mar. 2018. <http://www.gwern.net/Complement>. [80](#)
204. S. Brass and C. Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, May 2006. [166](#)
205. R. A. Brealey, S. C. Myers, and F. Allen. *Principles of Corporate Finance*. McGraw-Hill Irwin, 10th edition, 2011. [58](#), [73](#)
206. B. Brembs, K. Button, and M. Munafò. Deep impact: Unintended consequences of journal rank. *Frontiers in Human Neuroscience*, 7(291), June 2013. [9](#)
207. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW'10, pages 301–310, Feb. 2010. [199](#), [200](#)
208. C. A. Brewer. Color use guidelines for mapping and visualization. In A. M. Maceachren and D. R. F. Taylor, editors, *Visualization in Modern Cartography*, chapter 7, pages 123–147. Pergamon, Nov. 1994. [206](#)
209. E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. Ts'o. Disks for data centers. Technical report, Google, Inc, Feb. 2016. [350](#)
210. Brigham Young trace repository. No longer available via Brigham Young website, 201? Copy kindly supplied by Dror G. Feitelson. [146](#)
211. P. Brinch Hansen and R. House. The COBOL compiler for the Siemens 3003. *BIT*, 6(1):1–23, Mar. 1966. [102](#)
212. S. Broadbent. Font requirements for next generation air traffic management systems. Technical Report HRS/HSP-006-REP-01, European Organisation for the Safety of Air Navigation, 2000. [25](#)
213. G. W. Brock. *The U.S. Computer Industry: A Study of Market Power*. Ballinger Publishing Company, 1975. [82](#)
214. L. D. Brock and H. A. Goodman. Reliability analysis of the F-8 digital fly-by-wire system. NASA Contractor Report 163110, Dryden Flight Research Center, Oct. 1981. [137](#)
215. G. Bronevetsky and B. R. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, ICS'08, pages 155–164, June 2008. [156](#)
216. J. Brooke. SUS: A 'quick' and 'dirty' usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, editors, *Usability Evaluation in Industry*, chapter 21, pages 189–194. Taylor and Francis, June 1996. [354](#)
217. J. Brooke. SUS: A retrospective. *Journal of Usability Studies*, 8(2):29–40, Feb. 2013. [354](#)
218. R. Brooks. A model of human cognitive behavior in writing code for computer programs, vol I. Report AFOSR-TR-75-1084, Carnegie Mellon University, May 1975. [33](#)
219. F. P. Brooks, Jr. *The Mythical Man-Month*. Addison-Wesley, anniversary edition, 1995. [6](#), [132](#)
220. G. D. A. Brown, I. Neath, and N. Chater. A temporal ratio model of memory. *Psychological Review*, 114(3):539–576, 2007. [30](#)
221. J. Brunner and P. C. Austin. Inflation of Type I error rate in multiple regression when independent variables are measured with error. *The Canadian Journal of Statistics*, 37(1):33–46, Mar. 2009. [266](#)
222. M. Brysbaert, W. Fias, and M.-P. Noël. The Whorfian hypothesis and numerical cognition: Is 'twenty-four' processed in the same way as 'four-and-twenty'? *Cognition*, 66(1):51–77, Apr. 1998. [180](#)
223. I. Buchmann. *Batteries in a Portable World: A Handbook on rechargeable Batteries for Non-engineers*. Cadex Electronix Inc, third edition, 2011. [348](#)
224. J. B. Buckheit and D. L. Donoho. WaveLab and reproducible research. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and Statistics*, chapter 5, pages 55–81. Springer-Verlag, 1995. [8](#)
225. M. Budden, P. Hadavas, L. Hoffman, and C. Pretz. Generating valid  $4 \times 4$  correlation matrices. *Applied Mathematics E-Notes*, 7:53–59, 2007. [213](#)
226. D. V. Budescu, H.-H. Por, S. B. Broomell, and M. Smithson. The interpretation of IPCC probabilistic statements around the world. *Nature Climate Change*, 4:508–512, Apr. 2014. [142](#)
227. D. J. Buettner. *Designing an Optimal Software Intensive System Acquisition: A Game Theoretic Approach*. PhD thesis, University of Southern California, Sept. 2008. [115](#), [122](#), [131](#), [306](#), [311](#), [335](#)
228. M. Bullynck. What is an operating system? A historical investigation (1954–1964). HAL Id: halshs-01541602, HAL archives-ouvertes.fr, Aug. 2017. [101](#)

229. N. Bulnet and M. H. Halstead. Impurities found in algorithm implementations. Technical Report CSD-TR 111, Purdue University, Mar. 1974. [178](#)
230. J. S. Bunderson and K. M. Sutcliffe. Management team learning orientation and business unit performance. *Journal of Applied Psychology*, 88(3):552–560, June 2003. [69](#)
231. Bureau of labor statistics. website, July 2019. <https://www.bls.gov/ces>. [92](#)
232. K. P. Burnham and D. R. Anderson. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods and Research*, 33(2):261–304, Nov. 2004. [268](#)
233. Q. L. Burrell. A note on ageing in a library circulation model. *Journal of Documentation*, 41(2):100–115, 1985. [31](#)
234. R. P. L. Buse and W. R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, July 2010. [177](#)
235. J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. PhD thesis, Eindhoven University of Technology, Sept. 2013. [316](#)
236. J. Businge, A. Serebrenik, and M. van den Brand. Survival of Eclipse third-party plug-ins. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance*, ICSM’12, pages 368–377, Sept. 2012. [315](#)
237. R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, 1993. [147](#)
238. G. Butts and K. Linton. The joint confidence level paradox: A history of denial. In *NASA 2009 Cost Estimating Symposium*. NASA Center for Aerospace Information, Apr. 2009. [113](#)
239. B. Calder, D. Grunwald, and B. Zorn. Quantifying behavioral differences between C and C++ programs. *Journal of Programming Languages*, 2(4):313–351, 1995. [165](#)
240. E. G. Cale, L. L. Gremillion, and J. L. McKenney. Price/performance patterns of U.S. computer systems. *Communications of the ACM*, 22(4):225–233, Apr. 1979. [83](#)
241. J. Calhoun, C. Savoie, M. Randolph-Gips, and I. Bozkurt. Human reliability analysis in spaceflight applications. *Quality and Reliability Engineering International*, 29(6):869–882, Aug. 2013. [19](#)
242. A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC’15, pages 255–270, Aug. 2015. [177](#), [179](#)
243. C. F. Camerer and E. F. Johnson. The process-performance paradox in expert judgment: How can the experts know so much and predict so badly? In K. A. Ericsson and J. Smith, editors, *Towards a general theory of expertise: Prospects and limits*. Cambridge University Press, 1991. [35](#)
244. J. I. D. Campbell. On the relation between skilled performance of simple division and multiplication. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 23(5):1140–1159, 1997. [46](#)
245. M. Campbell-Kelly. *Foundations of Computer Programming in Britain (1945 - 1955)*. PhD thesis, Department of Mathematics and Computer Studies, Sunderland Polytechnic, June 1980. [96](#)
246. M. Campbell-Kelly. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. The MIT Press, Apr. 2004. [55](#)
247. M. Campbell-Kelly and D. D. Garcia-Swartz. Economic perspectives on the history of the computer time-sharing industry, 1965–1985. *IEEE Annals of the History of Computing*, 30(1):16–36, Jan.-Mar. 2008. [94](#)
248. M. Campbell-Kelly and D. D. Garcia-Swartz. Pragmatism not ideology: IBM’s love affair with open source software. SSRN Working Paper n. 1081613, Jan. 2008. [62](#)
249. M. Caneill and S. Zacchiroli. Debsources: Live and historical views on macro-level software evolution. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’14, pages 28:1–28:10, Sept. 2014. [105](#)
250. G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR’11, pages 143–152, May 2011. [228](#)
251. L. F. Capretz, P. Waychal, and J. Jia. Comparing popularity of testing careers among Canadian, Chinese, Indian students. In *IEEE/ACM 41st International Conference on Software Engineering*, ICSE-SEET, pages 258–259, May 2019. [98](#)
252. B. Caprile and P. Tonella. Nomen est omen: Analyzing the language of function identifiers. In *Proceedings of the 6<sup>th</sup> Working Conference on Reverse Engineering*, WCRE’99, pages 112–122, Oct. 1999. [175](#)
253. S. Carter-Thomas and E. Rowley-Jolivet. *If*-conditionals in medical discourse: From theory to disciplinary practice. *Journal of English for Academic Purposes*, 7(3):191–205, July 2008. [40](#)
254. E. Caspi. Empirical study of opportunities for bit-level specialization in word-based programs. Thesis (m.s.), University of California, Berkeley, 2000. [182](#)
255. D. Castelvecchi. The biggest mystery in mathematics: Shinichi Mochizuki and the impenetrable proof. *Nature*, 526(7572):178–181, Oct. 2015. [138](#)
256. J. P. Cavanagh. Relation between the immediate memory span and the memory search rate. *Psychological Review*, 79(6):525–530, 1972. [28](#)
257. M. Ceccato, M. Di Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19(4):1040–1074, 2014. [177](#)
258. C. Cecot and W. K. Viscusi. Judicial review of agency benefit-cost analysis. *George Mason Law Review*, 22(3):575–617, Nov. 2015. [137](#)
259. C. Cederström and P. Fleming. *Dead Man Working*. Zero books, 2012. [61](#)
260. D. Centola and A. Baronchelli. The spontaneous emergence of conventions: An experimental study of cultural evolution. *PNAS*, 112(7):1989–1994, Feb. 2015. [68](#)
261. D. Centola, J. Becker, D. Brackbill, and A. Baronchelli. Experimental evidence for tipping points in social convention. *Science*, 360(6393):1116–1119, June 2018. [68](#)
262. P. E. Ceruzzi. The early computers of Konrad Zuse, 1935 to 1945. *Annals of the History of Computing*, 3(3):241–262, July 1981. [1](#)
263. H. S. Cha. *Disrupting the Management Supply Chain: An Organizational Learning Model of IT Offshore Outsourcing*. PhD thesis, Faculty of the Committee on Business Administration, The University of Arizona, July 2007. [70](#)
264. F. Chandler, I. A. Heard, M. Presley, A. Burg, E. Midden, and P. Mongan. NASA human error analysis. Technical report, NASA Office of Safety and Mission Assurance, Sept. 2010. [19](#)
265. F. T. Chandler, Y. H. J. Chang, A. Mosleh, J. L. Marble, R. L. Boring, and D. I. Gertman. Human reliability analysis methods: Selection guidance for NASA. Nasa/osma technical report, NASA Headquarters Office of Safety and Mission Assurance, July 2006. [19](#)
266. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection—A survey. *ACM Computing Surveys*, 41(3):1–58, July 2009. [359](#), [360](#)
267. K. Chandrasekar. *High-Level Power Estimation and Optimization of DRAMs*. PhD thesis, Technische Universiteit Delft, Oct. 2014. [351](#)
268. A. C. Chang and P. Li. Is economics research replicable? Sixty published papers from thirteen journals say “usually not”. *Finance and Economics Discussion Series* 2015-083, Washington: Board of Governors of the Federal Reserve System, Sept. 2015. [9](#)
269. W. Chang. *R Graphics Cookbook*. O’Reilly, 2012. [203](#)
270. A. Chao. Estimating population size for sparse data in capture-recapture experiments. *Biometrics*, 45(2):427–438, June 1989. [93](#)
271. A. Chao, C.-H. Chiu, and L. Jost. Unifying species diversity, phylogenetic diversity, functional diversity, and related similarity and differentiation measures through Hill numbers. *Annual Review of Ecology, Evolution and Systematics*, 45(1):297–324, Nov. 2014. [86](#)
272. A. Chao, R. K. Colwell, C.-W. Lin, and N. J. Gotelli. Sufficient sampling for asymptotic minimum species richness estimators. *Ecology*, 90(4):1125–1133, Apr. 2009. [93](#)
273. A. Chao, S.-M. Lee, and S.-L. Jeng. Estimating population size for capture–recapture data when capture probabilities vary by time and individual animal. *Biometrics*, 48(1):201–216, Mar. 1992. [93](#)
274. A. Chao and C.-W. Lin. Nonparametric lower bounds for species richness and shared species richness under sampling without replacement. *Biometrics*, 68(3):912–921, Sept. 2012. [93](#)

275. A. Chao and M. C. K. Yang. Stopping rules and estimation for recapture debugging with unequal failure rates. *Biometrika*, 80(1):193–201, Mar. 1993. [162](#)
276. G. Charness and U. Gneezy. Strong evidence for gender differences in risk taking. *Journal of Economic Behavior & Organization*, 83(1):50–58, June 2012. [48](#)
277. W. G. Chase and K. A. Ericsson. Skill and working memory. In G. H. Bower, editor, *The Psychology of Learning and Motivation*, pages 1–58. Academic Press, 1982. [35](#)
278. N. Chater. *The Mind is Flat: The Illusion of Mental Depth and the Improvised Mind*. Allen Lane, Mar. 2018. [16](#)
279. P. D. Chatzoglou and L. A. Macaulay. Requirements capture and analysis : A survey of current practice. *Requirements Engineering*, 1(2):75–87, June 1996. [127](#)
280. D. D. Chen and G.-J. Ahn. Security analysis of x86 processor microcode. Thesis (B.Sc.), Arizona State University, Dec. 2014. [151](#)
281. T. Chen, Y. Chen, Q. Guo, O. Temam, T. Wu, and W. Hu. Statistical performance comparisons of computers. In *18th International Symposium on High Performance Computer Architecture*, HPCA, pages 1–12, Feb. 2012. [235](#), [236](#), [239](#)
282. Y. Chen, A. Groce, X. Fern, C. Zhang, W.-K. Wong, E. Eide, and J. Regehr. Taming compiler fuzzers. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI’13, pages 197–208, June 2013. [160](#), [297](#)
283. P. W. Cheng, K. J. Holyoak, R. E. Nisbett, and L. M. Oliver. Pragmatic versus syntactic approaches to training deductive reasoning. *Cognitive Psychology*, 18(3):293–328, July 1986. [35](#)
284. A. Chesson and G. Chamberlin. Survey-based measures of software investment in the UK. *Economic Trends* 627, Office for National Statistics, UK, Feb. 2006. [56](#)
285. R. N. Chesterman. Report of Queensland health payroll system commission of inquiry. Report, Queensland Government, Australia, July 2013. [113](#), [119](#)
286. H. Cheung and S. Kemper. Competing complexity metrics and adults' production of complex sentences. *Applied Psycholinguistics*, 13:53–76, 1992. [169](#)
287. R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980. [227](#)
288. J. Y. Chiao, A. R. Bordeaux, and N. Ambady. Mental representations of social status. *Cognition*, 93(2):B49–B57, Sept. 2004. [46](#)
289. J. J. Chilenski. An investigation of three forms of the modified condition decision coverage (MCDC) criterion. Final Report DOT/FAA/AR-01/18, U.S. Department of Transportation, Federal Aviation Administration, Apr. 2001. [161](#)
290. S. Chilton, J. Covey, M. Jones-Lee, G. Loomes, and H. Metcalf. Valuation of health benefits associated with reductions in air pollution. Technical report, Department for Environment, Food and Rural Affairs, May 2004. [142](#)
291. C.-H. Chiu, Y.-T. Wang, B. A. Walther, and A. Chao. An improved nonparametric lower bound of species richness via a modified Good-Turing frequency formula. *Biometrics*, 70(3):671–682, Sept. 2014. [93](#)
292. S. Christey and B. Martin. Buying into the bias: Why vulnerability statistics suck. blackhat USA 2013, July-Aug. 2013. [141](#)
293. T. Christie. The widespread and persistent myth that it is easier to multiply and divide with Hindu-Arabic numerals than with Roman ones. blog, Feb. 2017. <https://thonyc.wordpress.com/2017/02/10/the-widespread-and-persistent-myth-that-it-is-easier-to-multiply-and-divide-with-hindu-arabic-numerals-than-with-roman-ones/>. [96](#)
294. R. A. Chubon and M. R. Hester. An enhanced standard computer keyboard system for single-finger and typing-stick typing. *Journal of Rehabilitation Research and Development*, 25(4):17–24, Oct.-Dec. 1988. [86](#)
295. A. CIA. Analytic thinking and presentation for intelligence producers: Analysis training handbook. Technical report, Office of Training and Education, Central Intelligence Agency, Aug. 199? [202](#)
296. Z. J. Ciechanowicz and A. C. De Weever. The 'completeness' of the Pascal test suite. *Software—Practice and Experience*, 14(5):463–471, 1984. [162](#)
297. D. Citron. MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences. In *Proceedings of the 30th annual international symposium on Computer architecture*, ISCA’03, pages 52–61, June 2003. [346](#)
298. D. Citron and D. G. Feitelson. "look it up" or "do the math": An energy, area, and timing analysis of instruction reuse and memoization. Technical Report H-0196, IBM, Oct. 2003. [343](#), [344](#)
299. I. Ciupa, A. Pretschner, M. Oriol, A. Leitner, and B. Meyer. On the number and nature of faults found by random testing. *Software Testing, Verification and Reliability*, 21(1):3–28, Mar. 2011. [160](#)
300. H. H. Clark. *Understanding language*. Cambridge University Press, 1996. [163](#)
301. H. H. Clark and D. Wilkes-Gibbs. Referring as a collaborative process. *Cognition*, 22:1–39, 1986. [69](#)
302. A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. [297](#)
303. W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth Advanced Book Program, 1985. [203](#)
304. W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, Sept. 1984. [203](#)
305. J. Clune, J.-B. Mouret, and H. Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863, Jan. 2013. [185](#)
306. A. Coad. Investigating the exponential age distribution of firms. *Economics: The Open-Access, Open-Assessment E-Journal*, 4(2010-17):1–30, Mar. 2010. [97](#)
307. N. M. Coe. *The growth and locational dynamics of the UK computer services industry, 1981–1996*. PhD thesis, Department of Geography, University of Durham, 1996. [97](#)
308. J. Coelho and M. T. Valente. Why modern open source projects fail. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE’17, pages 186–196, Sept. 2017. [112](#)
309. J. Cohen. *Statistical Power Analysis for the Behavioural Sciences*. Routledge, second edition, 1988. [234](#), [236](#)
310. J. Cohen. The Earth is round ( $p < 0.05$ ). *American Psychologist*, 49(12):997–1003, 1994. [243](#)
311. J. Cohen, S. Teleki, and E. Brown. *Best Kept Secrets of Peer Code Review*. SmartBear Software, 2012. [357](#)
312. Z. Coker, S. Hasan, J. Overbey, M. Hafiz, and C. Kästner. Integers in C: An open invitation to security attacks? Technical Report CSSE14-01, Auburn University, Feb. 2014. [185](#)
313. M. Cokol, I. Iossifov, R. Rodriguez-Esteban, and A. Rzhetsky. How many scientific papers should be retracted? *European Molecular Biology Organization*, 8(5):422–423, Apr. 2007. [9](#)
314. M. Collard, A. Ruttle, B. Buchanan, and M. J. O’Brien. Population size and cultural evolution in nonindustrial food-producing societies. *PLoS ONE*, 8(9):e72628, Sept. 2013. [68](#)
315. C. Collberg, T. Proebsting, and A. M. Warren. Repeatability and benefaction in computer systems research-A study and a modest proposal. Technical Report TR 14-014, Department of Computer Science, University of Arizona, Feb. 2015. [9](#)
316. D. Comin and B. Hobijn. Cross-country technology adoption: making the theories face the facts. *Journal of Monetary Economics*, 51(1):39–83, 2004. [4](#)
317. C. Commeyne, A. Abran, and R. Djouab. Effort estimation with story points and cosmic function points - an industry case study. *Software Measurement News*, 21(1):25–36, 2016. [116](#), [117](#)
318. Committee of Public Accounts. HM revenue and customers: ASPIRE-re-competition of outsourced IT services. Technical Report Twenty-eighth Report of Session 2006-07, UK Parliament, June 2007. [89](#), [90](#), [120](#)
319. Comptroller General of the United States. Multiyear leasing and government-wide purchasing of automatic data processing equipment should result in significant savings. Technical Report B-115369, U.S. General Accounting Office, Apr. 1971. [94](#)
320. Comptroller General of the United States. Federal agencies' maintenance of computer programs: Expensive and undermanaged. Technical Report AFMD-81-25, U.S. General Accounting Office, Feb. 1981. [103](#)

321. S. Condon, M. Regardie, M. Stark, and S. Waligora. Cost and schedule estimation study report. Technical Report SEL-93-002, Goddard Space Flight Center, Nov. 1993. 114, 126
322. Foundations for evidence-based policymaking Act of 2018 H.R.4175, Jan. 2018. 115<sup>th</sup> Congress of the United States of America, 2<sup>nd</sup> session. 2
323. B. Conrad and M. Mitzenmacher. Power laws for monkeys typing randomly: The case of unequal probabilities. *IEEE Transactions on Information Theory*, 50(7):1403–1414, July 2004. 225
324. J. J. Cook and C. Zilles. A characterization of instruction-level error derating and its implications for error detection. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, DSN 2008, pages 482–491, June 2008. 264
325. K. Cook. Ubuntu security hardening statistics (amd64). <https://outflux.net/ubuntu/hardening/amd64>, July 2019. 92
326. P. Coombs. *IT Project Estimation: A Practical Guide to the Costing of Software*. Cambridge University Press, 2003. 117
327. A. Corazza, V. Maggio, and G. Scannello. Coherence of comments and method implementations: a dataset and an empirical investigation. *Software Quality Journal*, 26(2):751–777, June 2018. 177
328. J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it? Technical report, The Linux Foundation, Dec. 2010. 111
329. J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it. Technical report, The Linux Foundation, Mar. 2012. 264
330. M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2142–2151, Dec. 2014. 199
331. J. W. Cortada. *The Digital Flood: The Diffusion of Information Technology Across the U.S., Europe and Asia*. Oxford University Press, Sept. 2012. 4, 92
332. M. J. Cortese and M. M. Khanna. Age of acquisition predicts naming and lexical-decision performance above and beyond 22 other predictor variables: An analysis of 2,342 words. *The Quarterly Journal of Experimental Psychology*, 60(8):1072–1082, Aug. 2007. 175
333. L. Cosmides and J. Tooby. Evolutionary psychology: A primer. Technical report, Center for Evolutionary Psychology, University of California, Santa Barbara, 1998. 16, 39
334. D. L. Costa and M. E. Kahn. Changes in the value of life, 1940–1980. Working Paper No. 9396, National Bureau of Economic Research, USA, Dec. 2002. 142
335. V. Costan and S. Devadas. Intel SGX explained. In *Cryptology ePrint Archive: Report 2016/086*, Jan. 2016. 83
336. D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi. How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. *The Journal of Systems and Software*, 113(C):27–43, Mar. 2016. 140
337. J. D. Couger and M. A. Colter. *Maintenance Programming: Improving Productivity Through Motivation*. Prentice-Hall, Inc, 1985. 65
338. N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–185, 2001. 27
339. M. F. Cowlishaw. Decimal floating-point: Algorithm for computers. In *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pages 104–111, June 2003. 139
340. J. F. Coyle and G. D. Polksy. Acqui-hiring. *Duke Law Journal*, 63(2):281–346, Nov. 2013. 83
341. W. Crooymans, P. Pradhan, and S. Jansen. Exploring network modelling and strategy in the Dutch product software ecosystem. In *Proceedings of the International Conference on Software Business*, IC-SOB 2015, page ???, June 2015. 97
342. F. E. Croxton and R. E. Stryker. Bar charts versus circle diagrams. *Journal of the American Statistical Association*, 22(160):473–482, Dec. 1927. 202
343. J. Culver. The life cycle of a cpu. website: accessed 13 Jul 2017, 2010. <http://www.cpushack.com/life-cycle-of-cpu.html>. 232, 233
344. G. Cumming and R. Maillardet. Confidence intervals and replication: Where will the next mean fall? *Psychological Methods*, 11(3):217–227, 2006. 246
345. C. R. Cummins. *The interpretation and use of numerically-quantified expressions*. PhD thesis, Research Centre for English and Applied Linguistics, University of Cambridge, Nov. 2011. 45
346. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, Nov. 1988. 121
347. B. Curtis, S. B. Sheppard, and E. Kruesi. Evaluation of software life cycle data from the PAVE PAWS project. Technical Report RADC-TR-80-28, Rome Air Development Center, Griffiss Air Force Base, Mar. 1980. 124, 140, 151
348. M. A. Cusumano. Factory concepts and practices in software development: An historical overview. Working Paper #3095-89 BPS, Alfred P. Sloan School of Management, Dec. 1989. 131
349. M. A. Cusumano. Shifting economies: From craft production to flexible systems and software factories. Working Paper #3325-91/BPS, Alfred P. Sloan School of Management, Aug. 1991. 131
350. K. Cwalina and B. Abrams. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Addison-Wesley, 2006. 166
351. J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, Feb. 2006. 157
352. A. Damasio. *Self Comes to Mind: Constructing the Conscious Brain*. Vintage books, 2012. 16
353. M. Daneman and P. A. Carpenter. Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior*, 19(4):450–466, Aug. 1980. 172
354. M. Daneman and P. A. Carpenter. Individual differences in integrating information between and within sentences. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 9(4):561–584, 1983. 172
355. C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts. No country for old members: User lifecycle and linguistic change in online communities. In *Proceedings of the 22nd international conference on World Wide Web*, WWW 2013, pages 307–318, May 2013. 174
356. A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. CPU DB: Recording microprocessor history. *Communications of the ACM*, 55(4):55–63, Apr. 2012. 86, 196, 345
357. J. Darley and C. D. Batson. From Jerusalem to Jericho: A study of situational and dispositional variables in helping behavior. *Journal of Personality and Social Psychology*, 27(1):100–108, 1973. 20
358. P. A. David. Clio and the economics of QWERTY. *The American Economic Review*, 75(2):332–337, May 1985. 86
359. J. W. Davidson, J. R. Rabung, and D. B. Whalley. Relating static and dynamic machine code measurements. Technical Report CS-89-03, Department of Computer Science, University of Virginia, July 1989. 166
360. C. J. Davis. The spatial coding model of visual word identification. *Psychological Review*, 117(3):713–758, July 2010. 28, 163
361. S. J. Davis and B. S. de la Parra. Application flows. ???, ???(??):???, Apr. 2017. 98, 131
362. S. J. Davis, J. MacCrisken, and K. M. Murphy. Economic perspectives on software design: PC operating systems and platforms. Working Paper No. 8411, National Bureau of Economic Research, USA, Aug. 2001. 1
363. S. Dayal. Characterizing HEC storage systems at rest. Technical Report CMU-PDL-08-109, Parallel Data Laboratory, Carnegie Mellon University, July 2008. 225
364. R. de Blieck. *Empirical studies on the economic impact of trust*. PhD thesis, Erasmus Research Institute of Management, Rotterdam, May 2015. 66
365. S. De Deyne, S. Verheyen, E. Ameel, W. Vanpaemel, M. J. Dry, W. Voorspoels, and G. Storms. Exemplar by feature applicability matrices and other Dutch normative data for semantic concepts. *Behavior Research Methods*, 40(4):1030–1048, Nov. 2008. 38
366. A. D. de Groot. *Thought and Choice in Chess*. Amsterdam University Press, 2008. 35

367. J. L. de la Vara, M. Borg, K. Wnuk, and L. Moonen. An industrial survey of safety evidence change impact analysis practice. *IEEE Transactions on Software Engineering*, 42(12):1095–1117, Dec. 2016. 101, 152
368. B. B. de Mesquita, A. Smith, R. M. Siverson, and J. D. Morrow. *The Logic of Political Survival*. The MIT Press, 2005. 121
369. R. A. De Millo, R. J. Lipton, and A. J. Perllis. Social processes and proofs of theorems and programs. *Communications of the ACM*, 22(5):271–280, May 1979. 138, 243
370. A. B. de Oliveira, J.-C. Petkovich, T. Reidemeister, and S. Fischmeister. DataMill: Rigorous performance evaluation made easy. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE’13, pages 137–148, Apr. 2013. 353, 354, 355
371. C. B. De Soto, M. London, and S. Handel. Social reasoning and spatial paralogic. *Journal of Personality and Social Psychology*, 2(4):513–521, 1965. 41, 42
372. K. De Vogeeler. *La loi de convexité énergie-fréquence de la consommation des programmes : modélisation, thermosensibilité et applications*. PhD thesis, Informatique [cs] Telecom ParisTech, Sept. 2015. 349
373. K. De Vogeeler, G. Memmi, and P. Jouvelot. Parameter sensitivity analysis of the energy/frequency convexity rule for nanometer-scale application processors. In *eprint arXiv:cs.DS/1508.07740*, Aug. 2015. 348
374. G. de Wit. Firm size distributions: An overview of steady-state distributions resulting from firm dynamics models. Technical Report N200418, EIM Business and Policy Research, Jan. 2005. 97
375. I. J. Deary. *Intelligence: A Very Short Introduction*. Oxford University Press, 2001. 47
376. B. K. Debnath, M. F. Mokbel, and D. J. Lilja. Exploiting the impact of database system configuration parameters: A design of experiments approach. *IEEE Data Engineering Bulletin*, 31(1):3–10, Mar. 2008. 345
377. T. Debsources developers. Statistics | debian sources. <https://sources.debian.org/stats>, June 2019. 103
378. A. Decan, T. Mens, and M. Claes. An empirical comparison of dependency issues in OSS packaging ecosystems. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*, SANER 2017, pages 2–12, Feb. 2017. 106
379. A. Decan, T. Mens, M. Claes, and P. Grosjean. On the development and distribution of R packages: An empirical analysis of the R ecosystem. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW’15, page ???, Sept. 2015. 105
380. A. Decan, T. Mens, M. Claes, and P. Grosjean. When GitHub meets CRAN: An analysis of inter-repository package dependency problems. In *23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER’16, pages 493–504, Mar. 2016. 105
381. A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *15th International Conference on Mining Software Repositories*, MSR’18, pages 181–191, May 2018. 154, 155
382. Defence technical information center. Search page for DTIC reports, July 2016. <http://dsearch.dtic.mil>. 5
383. S. Dehaene. Symbols and quantities in parietal cortex: elements of a mathematical theory of number representation and manipulation. In P. Haggard, Y. Rossetti, and M. Kawato, editors, *Sensorimotor Foundations of Higher Cognition (Attention and Performance) XXII*, chapter 24, pages 527–574. Oxford University Press, Nov. 2007. 43
384. S. Dehaene. *Reading in the Brain: The Science and evolution of a human invention*. Viking, 2009. 15
385. S. Dehaene. *The Number Sense*. Oxford University Press, revised and updated edition, 2011. 41, 44
386. S. Dehaene, S. Bossini, and P. Giroux. The mental representation of parity and number magnitude. *Journal of Experimental Psychology: General*, 122(3):371–396, Sept. 1993. 19
387. S. Dehaene, E. Dupoux, and J. Mehler. Is numerical comparison digits? Analogical and symbolic effects in two-digit number comparisons. *Journal of Experimental Psychology: Human Perception and Performance*, 16(3):626–641, 1990. 44
388. S. Dehaene, V. Izard, E. Spelke, and P. Pica. Log or linear? Distinct intuitions of the number scale in Western and Amazonian indigenous cultures. *Science*, 320(5880):1217–1220, May 2008. 17, 44
389. S. M. Dekleva. The influence of the information systems development approach on maintenance. *MIS Quarterly*, 16(3):355–372, Sept. 1992. 134
390. R. T. DeLamarter. *Big Blue: IBM’s Use and Abuse of Power*. Pan Books, 1988. 70, 117
391. S. DellaVigna. Psychology and economics: Evidence from the field. Working Paper No. 13420, National Bureau of Economic Research, USA, Sept. 2007. 51
392. J. Demmel and Y. Hilda. Accurate floating point summation. Technical Report UCB/CSD-02-1180, University of California, Berkeley, May 2002. 137
393. Department of Defense. Military standard DOD-STD-2167 defense system software development. Standard DOD-STD-2167, U.S. Department of Defense, 1985. 123
394. Department of Defense. Standard practice system safety. Standard MIL-STD-882E, U.S. Department of Defense, May 2012. 142
395. G. Destefanis. Which programming language should a company use? A Twitter-based analysis. Technical Report CRIM-14/10-23-MODL, Computer Research Institute of Montréal, Oct. 2014. 103
396. S. Deutsch and M. H. Jørgensen. Studying the hidden costs of offshoring – the effect of psychic distance. Thesis (m.s.), Copenhagen Business School, Aug. 2014. 116
397. J. P. DeVale. *High Performance Robust Computer Systems*. PhD thesis, Electrical and Computer Engineering, Pittsburgh, Oct. 2001. 145
398. A. Di Franco, H. Guo, and C. Rubio-González. A comprehensive study of real-world numerical bug characteristics. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE’17, pages 509–519, Nov. 2017. 151
399. C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of Blue Waters. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN 2014, pages 610–621, June 2014. 156
400. M. Di Penta, L. Cerulo, and L. Aversano. The life and death of statically detected vulnerabilities: an empirical study. *Information and Software Technology*, 51(10):1469–1484, Oct. 2009. 143, 322, 323
401. T. F. Dickey. Programmer variability. *Proceedings of the IEEE*, 69(7):844–845, July 1981. 7
402. L. S. Dickstein. The effect of figure on syllogistic reasoning. *Memory & Cognition*, 6(1):76–83, 1978. 41
403. A. Diekmann. Not the first digit! Using Benford’s law to detect fraudulent scientific data. *Journal of Applied Statistics*, 34(3):321–329, Oct. 2007. 366
404. J. Dietrich, K. Jezek, and P. Brada. What Java developers know about compatibility, and why this matters. In *eprint arXiv:cs.SE/1408.2607v1*, Aug. 2014. 355, 357
405. C. DiMarco, G. Hirst, and M. Stede. The semantic and stylistic differentiation of synonyms and near-synonyms. In *AAAI Spring Symposium on Building Lexicons for Machine Translation*, pages 114–121, Mar. 1993. 214
406. A. Dinaburg. Bitsquatting: DNS hijacking without exploitation. Reference 2011-307, Raytheon Company, July 2011. 156
407. D. K. Dirlam. Most efficient chunk sizes. *Cognitive Psychology*, 3(2):355–359, Apr. 1972. 30
408. A. K. Dixit and R. S. Pindyck. *Investment under Uncertainty*. Princeton University Press, 1994. 59, 60, 313
409. H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2008, pages 71–82, Nov. 2008. 162
410. C. Domas. Breaking the x86 ISA. blackhat USA 2017, July 2017. 151
411. D. J. Dooling and R. E. Christiaansen. Episodic and semantic aspects of memory for prose. *Journal of Experimental Psychology: Human Learning and Memory*, 3(4):428–436, 1977. 171

412. J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS'99, pages 59–70, July 1999. 225
413. J. R. Doyle. Survey of time preference, delay discounting models. *Judgment and Decision Making*, 8(2):116–135, Mar. 2013. 51
414. G. Dréan. *The Computer Industry: Structure, economics, perspectives*. Gérard Dréan, english edition, 2012. 84
415. S. Drobisz, T. Mens, and R. Di Cosmo. A historical analysis of Debian package conflicts. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR'15, pages 212–223, June 2015. 105, 106
416. J. Duggan. Implementing a metapopulation Bass diffusion model using the R package deSolve. *The R Journal*, 9(1):153–163, June 2017. 335
417. R. I. M. Dunbar and R. Sosis. Optimising human community sizes. *Evolution and Human Behavior*, 39(1):106–111, Jan. 2018. 89, 90
418. J. R. Dunham and L. A. Lauterbach. An experiment in software reliability additional analyses using data from automated replications. NASA Contractor Report 178395, Research Triangle Institute, North Carolina, Jan. 1988. 147
419. J. R. Dunham and J. L. Pierce. An experiment in software reliability. NASA Contractor Report 172553, NASA Langley Research Center, Mar. 1986. 146, 147, 207
420. L. M. Dunn. *An Investigation of the Factors Affecting the Lifecycle Costs of COTS-Based Systems*. PhD thesis, School of Computing, University of Portsmouth, June 2011. 101
421. D. Dunning, C. Heath, and J. M. Suls. Flawed self-assessment: Implications for health, education, and the workplace. *Psychological Science in the Public Interest*, 5(3):69–106, Apr. 2004. 355
422. C. Dutang. CRAN task view: Probability distributions. website, June 2016. <http://CRAN.R-project.org/view=Distributions>. 214, 220
423. G. Dutilh, J. Annis, S. D. Brown, P. Cassey, N. J. Evans, R. P. P. P. Grasman, G. E. Hawkins, A. Heathcote, W. R. Holmes, A.-M. Kryptos, C. N. Kupitz, F. P. Leite, V. Lerche, Y.-S. Lin, G. D. Logan, T. J. Palmeri, J. J. Starns, J. S. Trueblood, L. van Maanen, D. van Ravenzwaaij, J. Vandekerckhove, I. Visser, A. Voss, C. N. White, T. V. Wiecki, J. Rieskamp, and C. Donkin. The quality of response time data inference: A blinded, collaborative assessment of the validity of cognitive models. *Psychonomic Bulletin & Review*, ???(??):???, Apr. 2018. 18
424. T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, Aug. 2006. 6, 338
425. R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen. Mining billions of AST nodes to study actual and potential usage of Java language features. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE'14, pages 779–790, May-June 2014. 182
426. P. C. Earley. Social loafing and collectivism: A comparison of the United States and the People's Republic of China. *Administrative Science Quarterly*, 34(4):565–581, Dec. 1989. 66
427. H. Ebbinghaus. *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie*. 1885. Translated by Henry A. Ruger and Clara E. Bussenius as Memory: A Contribution to Experimental Psychology (Teachers College, Columbia University, 1913). 31
428. A. Eckbreth, C. Saff, K. Connolly, N. Crawford, C. Eick, M. Goorsky, N. Kacena, D. Miller, R. Schafrik, D. Schmidt, D. Stein, M. Stroscio, G. Washington, and J. Zolper. Sustaining Air Force aging aircraft into the 21<sup>st</sup> century. Technical Report SAB-TR-11-01, United States Air Force Scientific Advisory Board, Aug. 2011. 89
429. Economist Data team. The changing US technology sector. Daily chart for April 21 2015 on The Economist webpage, Apr. 2015. As of Q1 2015, Sources: Thomson Reuters; awk scripts+R converted the data embedded in Javascript. 2, 3
430. EDB. Offensive security's exploit database archive. <https://www.exploit-db.com/>, Mar. 2018. 141
431. S. Eder, M. Junker, E. Jürgens, B. Hauptmann, R. Vaas, and K.-H. Prommer. How much does unused code matter for maintenance? In *34th International Conference on Software Engineering*, ICSE'12, pages 1102–1111, June 2012. 59
432. A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner. An empirical study on the effectiveness of security code review. In *Proceedings of the 5th international conference on Engineering Secure Software and Systems*, ESSoS'13, pages 197–212, Feb. 2013. 245, 246, 271, 279
433. M. A. Edwards and S. Roy. Academic research in the 21st century: Maintaining scientific integrity in a climate of perverse incentives and hypercompetition. *Environmental Engineering Science*, 34(1):51–61, Jan. 2017. 8
434. K. Ehrlich and P. N. Johnson-Laird. Spatial descriptions and referential continuity. *Journal of Verbal Learning and Verbal Behavior*, 21(3):296–306, June 1982. 173
435. S. G. Eick, C. R. Loader, M. D. Long, L. G. Votta, and S. V. Wiel. Estimating software fault content before coding. In *Proceedings of the 14th international conference on Software engineering*, ICSE'92, pages 59–65, May 1992. 159
436. P. Ein-Dor. Grosch's law re-revisited: CPU power and the cost of computation. *Communications of the ACM*, 28(2):142–151, Feb. 1985. 83
437. T. Eisensee and D. Strömberg. News droughts, news floods, and U.S. disaster relief. *The Quarterly Journal of Economics*, 122(2):693–728, May 2007. 142
438. K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai. The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering*, 28(5):494–509, Mar. 2002. 207
439. K. El Emam and A. G. Koru. A replicated survey of IT software project failures. *IEEE Software*, 25(5):84–90, Apr. 2008. 112
440. A. Elci. The dependence of operating system size upon allocatable resources. Technical Report 75-172, Department of Computer Science, Purdue University, Dec. 1975. 100
441. I. R. Elliott. Life cycle planning for a large mix of commercial systems. In B. Elkins and L. Hunt, editors, *Software Phenomenology – Working Papers of the Software Life Cycle Management Workshop*, chapter 10, pages 203–215. Computer Systems Command, United States Army, Aug. 1977. 134
442. J. Elliott, M. Hoemmen, and F. Mueller. Exploiting data representation for fault tolerance. In *eprint arXiv:cs.NA/1312.2333v1*, Dec. 2013. 137
443. N. C. Ellis and R. A. Hennelly. A bilingual word-length effect: Implications for intelligence testing and the relative ease of mental calculation in Welsh and English. *British Journal of Psychology*, 71:43–51, 1980. 27, 342
444. P. D. Ellis. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press, 2010. 234
445. R. Engbert, A. Nuthmann, E. M. Richter, and R. Kliegl. SWIFT: A dynamical model of saccade generation during reading. *Psychological Review*, 112(4):777–813, Apr. 2005. 25
446. J. Engblom. Why SpecInt95 should not be used to benchmark embedded systems tools. *ACM SIGPLAN Notices*, 34(7):96–103, July 1999. 7, 179
447. J. Ensign and D. K. Akaka. Defense acquisitions: DOD has paid billions in award and incentive fees regardless of acquisition outcomes. Technical Report GAO-06-66, United States Government Accountability Office, Dec. 2005. 120
448. N. L. Ensmenger. Letting the "computer boys" take over: Technology and the politics of organizational transformation. *International Review of Social History*, 48(S11):153–180, Dec. 2003. 122
449. Y.-H. Eom and H.-H. Jo. Generalized friendship paradox in complex networks: The case of scientific collaboration. In *eprint arXiv:cs.SI/1401.1458*, Apr. 2014. 90
450. D. M. Erceg-Hurn and V. M. Mirosevich. Modern robust statistical methods. *American Psychologist*, 63(7):591–601, Oct. 2008. 231
451. K. A. Ericsson and N. Charness. Expert performance. *American Psychologist*, 49(8):725–747, Aug. 1994. 35
452. K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, 100(3):363–406, 1993. also University of Colorado, Technical Report #91-06. 35
453. K. A. Ericsson and A. C. Lehmann. Expert and exceptional performance: Evidence of maximal adaption to task constraints. *Annual Review of Psychology*, 47:273–305, 1996. 35

454. K. Eriksson, D. H. Bailey, and D. C. Geary. The grammar of approximating number pairs. *Memory & Cognition*, 38(3):333–343, Apr. 2010. [44](#)
455. K. Eriksson, F. Jansson, and J. Sjöstrand. Bentley’s conjecture on popularity toplist turnover under random copying. *The Ramanujan Journal*, 23(1-3):371–396, Dec. 2010. [68](#)
456. L. Eshkevari, F. D. Santos, J. R. Cordy, and G. Antoniol. Are PHP applications ready for Hack? In *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering*, SANER 2015, pages 63–72, Mar. 2015. [185](#)
457. L. M. Eshkevari, V. Arnaoudova, M. Di Penta, R. Oliveto, Y.-G. Guéhéneuc, and G. Antoniol. An exploratory study of identifier renamings. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR’11, pages 33–42, May 2011. [118](#), [131](#)
458. H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS XVI, pages 319–332, Mar. 2011. [241](#)
459. W. K. Estes. *Classification and Cognition*. Oxford University Press, 1994. [37](#)
460. J. S. B. T. Evans, J. L. Barston, and P. Pollard. On the conflict between logic and belief in syllogistic reasoning. *Memory & Cognition*, 11(3):295–306, 1983. [41](#)
461. J. L. Eveleens and C. Verhoef. The rise and fall of the Chaos report figures. *IEEE Software*, 27(1):30–36, Jan. 2010. [112](#)
462. J. Eyolfson, L. Tan, and P. Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR’11, pages 153–162, May 2011. [112](#), [305](#), [324](#)
463. J. Eyolfson, L. Tan, and P. Lam. Correlations between bugginess and time-based commit characteristics. *Empirical Software Engineering*, 19(4):1009–1039, Aug. 2014. [325](#), [326](#), [327](#), [328](#)
464. Facebook. Facebook Inc. 2013 Form 10-K. website: accessed on 25 Feb 2017, 2014. <https://www.sec.gov/Archives/edgar/data/1326801/00013268011400007/fb-12312013x10k.htm>. [79](#)
465. Facebook. Facebook Inc. 2015 Form 10-K. website: accessed on 13 Feb 2017, 2016. <https://www.sec.gov/Archives/edgar/data/1326801/000132680116000043/fb-12312015x10k.htm>. [79](#)
466. R. Falk and C. Konold. Making sense of randomness: Implicit encoding as a basis for judgment. *Psychological Review*, 104(2):301–318, 1997. [46](#)
467. D. Fanelli. How many scientists fabricate and falsify research? A systematic review and meta-analysis of survey data. *PLoS ONE*, 4(5):e5738, May 2009. [9](#)
468. D. Fanelli. "Positive" results increase down the hierarchy of the sciences. *PLoS ONE*, 5(4):e10068, Apr. 2010. [9](#)
469. F. C. Fang, R. G. Steen, and A. Casadavall. Misconduct accounts for the majority of retracted scientific papers. *PNAS*, 109(42):17028–17033, Oct. 2012. [9](#)
470. M. Fang and M. Hafiz. Discovering buffer overflow vulnerabilities in the wild: An empirical study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’14, pages 23:1–23:10, Sept. 2014. [141](#)
471. L. Farr and B. Nanus. Factors that affect the cost of computer programming, volume I. Technical Documentary Report ESD-TDR-64-448, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, July 1964. [114](#)
472. L. Farr and H. J. Zagorski. Factors that affect the cost of computer programming, volume II: A quantitative analysis. Technical Documentary Report ESD-TDR-64-448, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Sept. 1964. [114](#)
473. J. Farrell and P. Klempner. Coordination and lock-in: Competition with switching costs and network effects. In M. Armstrong and R. H. Porter, editors, *Handbook of Industrial Organization*, Volume 3, chapter 31, pages 1967–2072. North-Holland, Oct. 2007. [89](#)
474. J. Farrell and C. Shapiro. Dynamic competition with switching costs. *RAND Journal of Economics*, 19(1):123–137, 1988. [90](#)
475. S. Farrell, M. J. Hurlstone, and S. Lewandowsky. Sequential dependencies in recall of sequences: Filling in the blanks. *Memory & Cognition*, 41(6):938–52, Aug. 2013. [30](#)
476. S. Farrell, K. Oberauer, M. Greaves, K. Pasiecznik, S. Lewandowsky, and C. Jarrold. A test of interference versus decay in working memory: Varying distraction within lists in a complex span task. *Journal of Memory and Language*, 90:66–87, Oct. 2016. [29](#)
477. FDA. General principles of software validation. Final guidance for industry and fda staff, U.S. Food and Drug Administration, Jan. 2002. [142](#)
478. Federal Register. *United States v. Adobe Systems, Inc., et al.; Proposed Final Judgment and Competitive Impact Statement*, 2010. 75 (No. 190; October 1), 24624. [98](#)
479. Federal Trade Commission. Dell computer corporation consent order, etc., in regard to alleged violation of sec. 5 of the federal trade commission act, docket c-3658. In P. C. Epperson, editor, *Federal Trade Commission decisions: Findings, opinions and orders volume 121*, pages 616–643. U.S. Government Printing Office, May 1996. [73](#)
480. D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2014. [101](#), [346](#), [366](#)
481. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, chapter 19, pages 337–360. Springer-Verlag, June 1995. [357](#)
482. S. L. Feld. Why your friends have more friends than you do. *The American Journal of Sociology*, 96(6):1464–1477, May 1991. [90](#)
483. J. Feldman. Minimization of boolean complexity in human concept learning. *Nature*, 407:630–633, Oct. 2000. [38](#)
484. J. Feldman. An algebra of human concept learning. *Journal of Mathematical Psychology*, 50(4):339–368, Aug. 2006. [38](#)
485. M. Felici. *Observational Models of Requirements Evolution*. PhD thesis, School of Informatics University of Edinburgh, 2004. [126](#), [127](#)
486. S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: Probabilistic soft error reliability on the cheap. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS’10, pages 385–396, Mar. 2010. [156](#)
487. N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause. On the effectiveness of early life cycle defect prediction with Bayesian nets. *Empirical Software Engineering*, 13(5):499–537, Oct. 2008. [271](#), [272](#)
488. D. V. Ferens and D. S. Christensen. Calibrating software cost models to Department of Defense databases-A review of ten studies. *ISPA Journal of Parametrics*, XVIII(2):55–74, Nov. 1998. [115](#)
489. C. J. Ferguson and M. Heene. A vast graveyard of undead theories publication bias and psychological science’s aversion to the Null. *Perspectives on Psychological Science*, 7(6):555–561, Nov. 2012. [9](#)
490. P. Fernández. Valuing real options: Frequently made errors. SSRN Working Paper n. 274855, Instituto de Estudios Superiores de la Empresa, Madrid, June 2001. [61](#)
491. L. Ferrand, M. Brysbaert, E. Keuleers, B. New, P. Bonin, A. Méot, M. Augustinova, and C. Pallier. Comparing word processing times in naming, lexical decision, and progressive demasking: evidence from Chronolex. *frontiers in Psychology*, 2(306), Nov. 2011. [175](#)
492. S. Ferson, J. O’Rawe, A. Antonenko, J. Siegrist, J. Mickley, C. C. Luhmann, K. Sentz, and A. M. Finkel. Natural language of uncertainty: numeric hedge words. *International Journal of Approximate Reasoning*, 57:19–39, Feb. 2015. [45](#), [46](#)
493. R. G. Fichman and C. F. Kemerer. Incentive compatibility and systematic software reuse. *Journal of Systems and Software*, 57(1):45–60, Apr. 2001. [118](#)
494. A. Filippin and P. Crosetto. A reconsideration of gender differences in risk attitudes. IZA DP No. 8184, The Institute for the Study of Labor, Bonn, May 2014. [48](#)
495. C. J. Fillmore. Topics in lexical semantics. In R. W. Cole, editor, *Current Issues in Linguistic Theory*, pages 76–138. Indiana University Press, 1977. [39](#)
496. Financial Accounting Standards Board. Statement of financial accounting standards no. 86. Technical report, Financial Accounting Foundation, Aug. 1985. [75](#)

497. M. Finifter. Towards evidence-based assessment of factors contributing to the introduction and detection of software vulnerabilities. Technical Report UCB/EECS-2013-49, Electrical Engineering and Computer Sciences, University of California at Berkeley, May 2013. [158](#)
498. M. Finifter, D. Akhawe, and D. Wagner. An empirical study of vulnerability rewards programs. In *Proceedings of the 22nd USENIX conference on Security*, SEC'13, pages 273–288, Aug. 2013. [143](#)
499. E. Fischer. The evolution of character codes, 1874–1968. Nov. 2002. [96](#)
500. D. A. Fisher. A common programming language for the Department of Defense – background and technical requirements. PAPER P-1191, Institute for Defense Analyses, Science and Technology Division, June 1976. [102](#)
501. J. Fisher and R. A. Hinde. The opening of milk bottles by birds. *British Birds*, 42(11):347–357, 1949. [68](#)
502. J. C. Fisher and R. H. Pry. A simple substitution model of technological change. *Technological Forecasting & Social Change*, 3:75–88, Apr. 1971–1972. [77](#)
503. P. Flajolet, P. Dumas, and V. Puyhaubert. Some exactly solvable models of urn process theory. In P. Chassaing, editor, *Proceedings of Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities*, pages 59–118, 2006. [92](#)
504. K. Flamm. *Targeting the Computer*. The Brookings Institution, Washington, D.C., 1987. [5, 94](#)
505. K. Flamm. *Creating the Computer*. The Brookings Institution, Washington, D.C., 1988. [1](#)
506. K. Flamm. Measuring Moore's law: Evidence from price, cost, and quality indexes. Working Paper No. 24553, National Bureau of Economic Research, USA, Apr. 2018. [4](#)
507. D. Flater. Estimation of uncertainty in application profiles. NIST TN.1826, National Institute of Standards and Technology, Apr. 2014. [354](#)
508. D. Flater. Screening for factors affecting application performance in profiling measurements. NIST Technical Note 1855, National Institute of Standards and Technology, Oct. 2014. [351](#)
509. D. Flater and W. F. Guthrie. A case study of performance degradation attributable to run-time bounds checks on C++ vector access. *Journal of Research of the National Institute of Standards and Technology*, 118(012):260–279, May 2013. [182, 274, 276](#)
510. P. J. Fleming and J. J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, Mar. 1986. [346](#)
511. J. I. Flombaum, J. A. Junge, and M. D. Hauser. Rhesus monkeys (*macaca mulatta*) spontaneously compute addition operations over large numbers. *Cognition*, 97(3):315–325, Oct. 2005. [44](#)
512. B. Floyd, T. Santander, and W. Weimer. Decoding the representation of code in the brain: An fMRI study of code review and expertise. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE'17, pages 175–186, May 2017. [163](#)
513. B. Flyvbjerg. How planners deal with uncomfortable knowledge: The dubious ethics of the American Planning Association. *Cities*, 32:157–163, June 2013. [113](#)
514. B. Flyvbjerg, M. S. Holm, and S. L. Buhl. Underestimating costs in public works projects: Error or lie? *Journal of the American Planning Association*, 68(3):279–295, June 2002. [113](#)
515. J. Fodor. *The Modularity of Mind: An Essay on Faculty Psychology*. MIT Press, 1983. [16](#)
516. R. A. Foley. An evolutionary and chronological framework for human social behaviour. *Proceedings of the British Academy*, 88:95–117, 1996. [15](#)
517. P. Fonseca, K. Zhang, X. Wang, and A. Krishnamurthy. An empirical study on the correctness of formally verified distributed systems. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys'17, pages 328–343, Apr. 2017. [138](#)
518. R. E. Fontana Jr. and G. M. Decad. Moore's law realities for recording systems and memory storage components: HDD, tape, NAND, and optical. *AIP Advances*, 8(5):056506, May 2018. [88](#)
519. C. E. Ford and S. A. Thompson. Conditionals in discourse: A text-based study from English. In E. C. Traugott, A. T. Meulen, J. S. Reilly, and C. A. Furguson, editors, *On Conditionals*, chapter 18, pages 353–372. Cambridge University Press, 1986. [40](#)
520. J. Förster, E. Higgins, and A. T. Bianco. Speed/accuracy decisions in task performance: Built-in trade-off or separate strategic concerns? *Organizational Behavior and Human Decision Processes*, 90(1):148–164, Jan. 2003. [20](#)
521. J. Fowkes and C. Sutton. Parameter-free probabilistic API mining across GitHub. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 254–265, Nov. 2016. [332](#)
522. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. [6](#)
523. W. B. Frakes, C. J. Fox, and B. A. Nejmeh. *Software Engineering in the Unix/C Environment*. Prentice-Hall, Inc, 1991. [166](#)
524. S. Frederick, G. Loewenstein, and T. O'Donoghue. Time discounting: A critical review. *Journal of Economic Literature*, 40(2):351–401, June 2002. [164](#)
525. D. P. Freedman and G. M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews*. Dorset House Publishing, 1990. [159](#)
526. P. A. Freund and N. Kasten. How smart do you think you are? A meta-analysis on the validity of self-estimates of cognitive ability. *Psychological Bulletin*, 138(2):296–321, Mar. 2011. [17](#)
527. A. Frumusanu. The Samsung Exynos 7420 deep dive - Inside a modern 14nm SoC. website, June 2015. <http://www.anandtech.com/show/9330/exynos-7420-deep-dive/5>. [348, 349](#)
528. W.-T. Fu and W. D. Gray. Memory versus perceptual-motor trade-offs in a blocks world task. In *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*, pages 154–159. Erlbaum, 2000. [21](#)
529. Y. Funami and M. H. Halstead. A software physics analysis of akiyama's debugging data. Technical Report CSD-TR 144, Purdue University, May 1975. [178](#)
530. B. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4):1–53, June 2010. [358](#)
531. C. A. Furia. Bayesian statistics in software engineering: Practical guide and case studies. In *eprint arXiv:cs.SE/1608.06865*, Aug. 2016. [124, 232](#)
532. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. Statistical semantics: Analysis of the potential performance of keyword information systems. *The Bell System Technical Journal*, 62(6):1753–1805, July-Aug. 1983. [95](#)
533. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, Nov. 1987. [95](#)
534. T. Futagami, M. Itoh, Y. Miura, F. Mitsuhashi, H. Nishiyama, M. Shukuguchi, N. Tachi, K. Toyama, H. Obata, Y. Oozumi, T. Shimizu, and S. Takeichi. *ESCR Embedded System development Coding Reference guide [C Language Edition]*. Information-technology Promotion Agency, Japan, 2.0 edition, 2017. [166](#)
535. R. Futrell, K. Mahowald, and E. Gibson. Large-scale evidence of dependency length minimization in 37 languages. *PNAS*, 112(33):10336–10341, Aug. 2015. [169](#)
536. M. T. Gailliot and R. F. Baumeister. The physiology of willpower: Linking blood glucose to self-control. *Personality and Social Psychology Review*, 11(4):303–327, Nov. 2007. [52](#)
537. W. A. Gale. Good-Turing smoothing without tears. Technical Report 94.5, AT&T Bell Laboratories, Aug. 1994. [363](#)
538. K. Gallaba, A. Mesbah, and I. Beschastnikh. Don't call us, we'll call you: Characterizing callbacks in JavaScript. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM'15, pages 247–256, Oct. 2015. [184, 185](#)
539. C. R. Gallistel, S. Fairhurst, and P. Balsam. The learning curve: Implications of a quantitative analysis. *PNAS*, 101(36):13124–13131, Sept. 2004. [32](#)
540. C. R. Gallistel, M. Krishan, Y. Liu, R. Miller, and P. E. Latham. The perception of probability. *Psychological Review*, 121(1):96–123, Apr. 2014. [46](#)

541. T. J. Gandomani, K. T. Wei, and A. K. Binhamid. A case study research on software cost estimation using experts' estimates, Wideband Delphi, and Planning Poker technique. *International Journal of Software Engineering and Its Applications*, 8(11):173–182, Apr. 2014. 251
542. A. Gandy. *The entry of established electronics companies into the early computer industry in the UK and USA*. PhD thesis, London School of Economics and Political Science, 1992. 1, 101
543. J. D. Gannon. An experimental evaluation of data type conversions. *Communications of the ACM*, 20(8):584–595, Aug. 1977. 167
544. Z. Gao, Y. Liang, M. B. Cohen, A. M. Memon, and Z. Wang. Making system user interactive tests repeatable: When and what should we control? In *Proceedings of the 37th International Conference on Software Engineering*, ICSE’15, pages 55–65, June 2015. 161
545. M. K. Gardner, E. Z. Rothkopf, R. Lapan, and T. Laferty. The word frequency effect in lexical decision: Finding a frequency-based component. *Memory & Cognition*, 15(1):24–28, 1987. 176
546. M. R. Garman. The generalizability of private sector research on software project management in two USAF organizations: An exploratory study. Thesis (m.s.), Air Force Institute of Technology, U.S.A., Mar. 2003. 110
547. R. Garner and F. R. Dill. The legendary IBM 1401 data processing system. *IEEE Solid-State Circuits Magazine*, 2(1):28–39, Jan. 2010. 94
548. Gartner. Worldwide smartphone sales. [https://en.wikipedia.org/wiki/Mobile\\_operating\\_system](https://en.wikipedia.org/wiki/Mobile_operating_system), July 2017. 3, 82
549. J. Gascoigne. Introducing open salaries at buffer our transparent formula and all individual salaries buffer. website, Dec. 2013. <https://open.buffer.com/introducing-open-salaries-at-buffer-including-our-transparent-formula-and-all-individual-salaries/>. 65
550. D. C. Gause and G. M. Weinberg. *Exploring Requirements: Quality before design*. Dorset House Publishing, 1989. 122
551. J. E. Gayek, L. G. Long, K. D. Bell, R. M. Hsu, and R. K. Larson. Software cost and productivity model. Technical Report ATR-2004(8311)-1, Aerospace Corporation, Feb. 2004. 75
552. Gcc releases. Vendor website, July 2019. <https://gcc.gnu.org/releases.html>. 85, 106
553. Y. Ge and B. Xu. Dynamic staffing and rescheduling in software project management: A hybrid approach. *PLoS ONE*, 11(6):e0157104, June 2016. 121
554. Y. Geffen and S. Maoz. On method ordering. In *IEEE 24th International Conference on Program Comprehension*, ICPC’16, page ???, May 2016. 186
555. W. Gellerich, M. Kosiol, and E. Ploedereder. Where does GOTO go to? In *Reliable Software Technology – Ada-Europe 1996*, volume 1088 of *LNCS*, pages 385–395. Springer, 1996. 183
556. S. A. Gelman and E. M. Markman. Categories and induction in young children. *Cognition*, 23:183–209, 1986. 36
557. D. Gentner and S. Goldin-Meadow. *Language In Mind: Advances in the Study of Language and Thought*. MIT Press, 2003. 180
558. S. L. Gerhart and L. Yelowitz. Observations of fallibility in applications of modern programming methodologies. *IEEE Transactions on Software Engineering*, SE-2(3):195–207, Sept. 1976. 138
559. M. Gerlach, B. Farb, W. Revelle, and L. A. N. Amaral. A robust data-driven approach identifies four personality types across four large data sets. *Nature Human Behaviour*, 2(10):735–742, Sept. 2018. 47
560. D. M. German, B. Adams, and A. E. Hassan. Continuously mining distributed version control systems: An empirical study of how Linux uses git. *Empirical Software Engineering*, 21(1):260–299, Feb. 2016. 165
561. D. M. German and J. M. González-Barahona. An empirical study of the reuse of software licensed under the GNU general public license. In *The 5th International Conference on Open Source Systems*, OSS 2009, pages 185–198, June 2009. 62
562. D. M. German, Y. Manabe, and K. Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE’10, pages 437–446, Apr. 2010. 63
563. E. H. Gibbs, G. A. Munroe, A. M. Zeman, and C. T. Cottingham. JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated and the general public, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, case no. 1-05-CV-039231, filing #g-43414. Opinion, Superior court of the state of California for the county of Santa Clara, 2012. 345
564. E. Gibson. Linguistic complexity: locality of syntactic dependencies. *Cognition*, 68(1):1–76, Aug. 1998. 169
565. E. Gibson and J. Thomas. Memory limitations and structured forgetting: The perception of complex ungrammatical sentences as grammatical. *Language and Cognitive Processes*, 14(3):225–248, 1999. 173
566. G. Gigerenzer. Striking a blow for sanity in theories of rationality. In M. Augier and J. G. March, editors, *Models of a man: Essays in memory of Herbert A. Simon*, pages 389–409. MIT Press, May 2004. 48
567. G. Gigerenzer. *Rationality for Mortals-How People cope with Uncertainty*. Oxford University Press, 2008. 39, 243
568. G. Gigerenzer, W. Gaissmaier, E. Kurz-Milcke, L. M. Schwartz, and S. Woloshin. Helping doctors and patients make sense of health statistics. *Psychological Science in the Public Interest*, 8(2):53–96, Apr. 2008. 209
569. G. Gigerenzer, S. Krauss, and O. Vitouch. The null ritual: What you always wanted to know about significance testing but were afraid to ask. In D. Kaplan, editor, *The Sage handbook of quantitative methodology for the social sciences*, chapter 21, pages 391–408. SAGE Publications, Inc, 2004. 244
570. G. Gigerenzer, P. M. Todd, and The ABC Research Group. *Simple Heuristics That Make Us Smart*. Oxford University Press, 1999. 16, 39, 48
571. B. Gilchrist and R. E. Weber. Employment of trained computer personnel—A quantitative survey. In *Proceedings of the Spring Joint Computer Conference*, AFIPS’72, pages 641–648, May 1972. 98
572. J. Gimpel. Software that checks software: The impact of PC-lint. *IEEE Software*, 31(1):15–19, Jan.-Feb. 2014. 131
573. V. Girotto, A. Mazzocco, and A. Tasso. The effect of premise order on conditional reasoning: a test of the mental model theory. *Cognition*, 63:1–28, 1997. 41
574. M. Givon, V. Mahajan, and E. Muller. Software piracy: Estimation of lost sales and the impact on software diffusion. *Journal of Marketing*, 59(1):29–37, Jan. 1995. 78, 310
575. T. J. Gauthier. Computer time sharing: Its origins and development. *Computers and Automation*, 16(10):23–27, Oct. 1967. 1
576. A. Glenberg. Few believe the world is flat: How embodiment is changing the scientific understanding of cognition. *Canadian Journal of Experimental Psychology*, 69(2):165–171, June 2015. 18
577. F. Gobet. *Understanding Expertise: A Multi-disciplinary Approach*. Palgrave, 2016. 35
578. D. R. Godden and A. D. Baddeley. Context-dependent memory in two natural environments: On land and underwater. *British Journal of Psychology*, 66(3):325–331, 1975. 30
579. M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *16<sup>th</sup> International Conference on Software Maintenance*, pages 131–142, Oct. 2000. 269
580. A. L. Goel. An experimental investigation into software reliability. Final Technical Report RADC-TR-88-213, CASE Center, Syracuse University, Oct. 1988. 152
581. M. Goeminne and T. Mens. Towards a survival analysis of database framework usage in Java projects. In *2015 IEEE 31st International Conference on Software Maintenance and Evolution*, ICSME 2015, pages 551–556, Sept.-Oct. 2015. 322, 323
582. S. S. Gokhale and R. E. Mullen. The marginal value of increased testing: An empirical analysis using four code coverage measures. *Journal of the Brazilian Computer Society*, 12(3):13–30, Dec. 2006. 162
583. M. M. Gold. A methodology for evaluating time-shared computer system usage. Technical report, Carnegie Mellon University, Aug. 1967. 106
584. L. R. Goldberg. An alternative "description of personality": The big-five factor structure. *Journal of Personality and Social Psychology*, 59(6):1216–12297, 1990. 47

585. L. R. Goldberg, J. A. Johnson, H. W. Eber, R. Hogan, M. C. Ashton, C. R. Cloninger, and H. G. Gough. The international personality item pool and the future of public-domain personality measures. *Journal of Research in Personality*, 40(1):84–96, 2006. 47
586. M. S. Goldberg and A. Touw. Statistical methods for learning curves and cost analysis. Technical Report CIMD0006870.A3/1Rev, The CNA Corporation, Mar. 2003. 69
587. H. H. Goldstine and J. von Neumann. Planning and coding of problems for an electronic computing instrument. Technical Report Part II, Volume 1-3, Institute for Advanced Study, Princeton, Apr. 1947. 122
588. P. Golle. Revisiting the uniqueness of simple demographics in the US population. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES’06, pages 77–80, Oct. 2006. 358
589. R. W. Gomulkiewicz. Enforcement of open source software licenses: The MDY trio’s inconvenient complications. *Yale Journal of Law and Technology*, 106(??):???, Apr. 2011. 64
590. R. Gonzalez and G. Wu. On the shape of the probability weighting function. *Cognitive Psychology*, 38(1):129–166, Feb. 1999. 46
591. J. M. González-Barahona, G. Robles, I. Herráiz, and F. Ortega. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process*, 26(7):589–612, July 2014. 199, 236, 297, 312
592. B. H. Good, Y.-A. de Montjoye, and A. Clauset. The performance of modularity maximization in practical contexts. In *eprint arXiv:physics.data-an/0910.0165v2*, Apr. 2010. 228
593. J. Goodman. Lessons learned from seven Space Shuttle missions. NASA Contractor Report CR-2007-213697, Lyndon B. Johnson Space Center, Jan. 2007. 138
594. P. Goodridge, J. Haskel, and G. Wallis. Estimating UK investment in intangible assets and intellectual property rights. Technical Report No. 2014/36, Intellectual Property Office, UK government, Sept. 2014. 3, 4
595. Google books ngram dataset. website, 2015. <http://storage.googleapis.com/books/ngrams/books/datasetsv2.html>. 365
596. A. Gopal and B. R. Koka. The role of contracts on quality and returns to quality in offshore software development outsourcing. *Decision Sciences*, 41(3):491–516, Aug. 2010. 120
597. R. Gopinath, A. Alipour, I. Ahmed, C. Jensen, and A. Groce. How hard does mutation analysis have to be, anyway? In *IEEE 26th International Symposium on Software Reliability Engineering*, ISSRE 2015, pages 216–227, Nov. 2015. 162, 213
598. R. Gopinath, C. Jensen, and A. Groce. Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE’14, pages 72–82, June 2014. 161, 162, 284
599. R. Gopinath, C. Jensen, and A. Groce. Mutations: How close are they to real faults? In *IEEE 25th International Symposium on Software Reliability Engineering*, ISSRE’14, pages 189–200, Nov. 2014. 154, 162
600. R. D. Gordon and M. H. Halstead. An experiment comparing Fortran programming times with the software physics hypothesis. Technical Report TR 167, Purdue University, Oct. 1975. 178
601. R. J. Gordon. The postwar evolution of computer prices. Working Paper No. 2227, National Bureau of Economic Research, USA, Apr. 1987. 3
602. M. Gottscho. ViPZonE: Exploiting DRAM power variability for energy savings in Linux x86-64. Thesis (m.s.), Electrical Engineering, UCLA, Mar. 2014. 351
603. M. Gottscho, A. A. Kagalwalla, and P. Gupta. Power variability in contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(12):37–40, June 2012. 350
604. S. Götz, T. Ilsche, J. Cardoso, J. Spillner, U. Aßmann, W. Nagel, and A. Schill. Energy-efficient data processing at sweet spot frequencies. In *OTM 2014 Workshops*, pages 154–171, Apr. 2014. 348
605. G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR’14, pages 368–371, May 2014. 195
606. G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator’s perspective. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368, May 2015. 200, 201
607. K. Goševa-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, ISSRE’05, pages 43–52, Nov. 2005. 226
608. E. M. Grabbe, S. Ramo, and D. E. Wooldridge. *Handbook of Automation, Computation, and Control, Volume 2: Computers and Data Processing*. John Wiley & Sons, Inc, 1959. 102
609. P. Grady. *Termination of the SIREN ICT project*. Grant Thornton UK LLP, June 2014. 123
610. R. B. Grady and D. L. Caswell. *Software Metrics: Establishing a company-wide program*. Prentice-Hall, Inc, 1987. 140
611. A. C. Graesser, N. L. Hoffman, and L. F. Clark. Structural components of reading time. *Journal of Verbal Learning and Verbal Behavior*, 19(2):135–151, Apr. 1980. 173
612. A. C. Graesser, S. B. Woll, D. J. Kowalski, and D. A. Smith. Memory for typical and atypical actions in scripted activities. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):503–515, June 1980. 171
613. S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière. Auto-tuning for floating-point precision with discrete stochastic arithmetic. HAL Id: hal-01331917, HAL archives-ouvertes.fr, June 2016. 139
614. E. E. Grant and H. Sackman. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Transactions on Human Factors in Electronics*, 8(1):33–48, Mar. 1967. 7, 52, 253
615. Graphviz-graph visualization software. website, 2015. <http://www.graphviz.org>. 200
616. C. A. Graver, W. M. Carriere, E. E. Balkovich, and R. Thibodeau. Cost reporting elements and activity cost tradeoffs for defense system software (study results). Technical Report ESD-TR-77-262, Vol. 1, General Research Corporation, May 1977. 126
617. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation & Assessment in Software Engineering 2011*, EASE 2011, pages 96–103, Apr. 2011. 358
618. J. Gray, C. Nyberg, M. Shah, and N. Govindaraju. Sort benchmark. <http://sortbenchmark.org>, July 2014. 71, 346
619. W. D. Gray, C. R. Sims, W.-T. Fu, and M. J. Schoelles. The soft constraints hypothesis: A rational analysis approach to resource allocation for interactive behavior. *Psychological Review*, 113(3):461–482, 2006. 21
620. M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale Java open source code repository. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’10, pages 11:1–11:10, Sept. 2010. 179
621. J. H. Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. In J. H. Greenberg, editor, *Universals of Language*, chapter 5, pages 58–90. MIT Press, 1963. 40
622. H. I. Greenfield. An economist looks at data processing. *Computers and Automation*, 6(10):18–23, Oct. 1957. 94
623. S. Greenstein. Did computer technology diffuse quickly?: Best and average practice in mainframe computers, 1968–1983. Working Paper No. 4647, National Bureau of Economic Research, USA, Feb. 1994. 89
624. C. Gregg and K. Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, pages 134–144, Apr. 2011. 341
625. P. Grice. *Studies in the Way of Words*. Harvard University Press, 1989. 163
626. D. A. Grier. The ENIAC, the verb "to program" and the emergence of digital computers. *IEEE Annals of the History of Computing*, 18(I):51–55, 1996. 1
627. D. A. Grier. *When Computers were Human*. Princeton University Press, 2005. 1, 82

628. S. Grimstad and M. Jørgensen. Inconsistency of expert judgement-based estimates of software development effort. *Journal of Systems and Software*, 80(11):1770–1777, Nov. 2007. 113
629. E. Grochowski and R. E. Fontana, Jr. Future technology challenges for NAND flash and HDD products. Flash Memory Summit 2012, Santa Clara, CA, July 2012. 296
630. U. Grömping. Relative importance for linear regression in R: The package *relaimpo*. *Journal of Statistical Software*, 17(1):1–27, Sept. 2006. 287
631. E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis. The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS ONE*, 7(6):e38234, June 2012. 139
632. H. R. J. Grosch. High speed arithmetic: The digital computer as a research tool. *Journal of the Optical Society of America*, 43(4):306–310, Apr. 1953. 1
633. A. S. Grove. *Only the Paranoid Survive: How to Exploit the Crisis Points That Challenge Every Company and Career*. HarperCollins-Busines, Apr. 1988. 82
634. A. Grubler and N. Nakićenović. Long waves, technology diffusion, and substitution. Technical Report RP-91-17, International Institute for Applied Systems Analysis Laxenburg, Austria, Oct. 1991. 2
635. W. Gruhl. Lessons learned cost/schedule assessment guide. Slides of talk, July 199? 195, 196
636. M. Gubler. *Protean and boundaryless career orientations - an empirical study of IT professionals in Europe*. PhD thesis, Loughborough University, July 2011. 64
637. T. Gue. Triggering infection: Distribution and derivative works under the GNU general public license. *Journal of Law, Technology & Policy*, 2012(1):95–140, 2012. 62
638. L. Guerrouj, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol. An experimental investigation on the effects of context on source code identifiers splitting and expansion. *Empirical Software Engineering*, 19(6):1706–1753, Dec. 2014. 342
639. H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing*, SOCC’14, pages 1–14, Nov. 2014. 256
640. H. S. Gunawi, C. Rubio-González, A. C. Arpacı-Dusseau, R. H. Arpacı-Dusseau, and B. L. Liblit. EIO: Error handling is Occasionally correct. In *Proceedings of the 6<sup>th</sup> USENIX Conference on File and Storage Technologies*, pages 207–222, Feb. 2008. 152
641. N. J. Gunther. A simple capacity model of massively parallel transaction systems. In *Proceedings of 19th International CMG Conference*, pages 1035–1044, Dec. 1993. 205
642. N. J. Gunther. *Analysing Computer System Performance with Perl::PDQ*. Springer-Verlag, 2005. 205, 346
643. J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wąsowski. Variability-aware performance prediction: A statistical learning approach. In *IEEE/ACM 28th International Conference on Automated Software Engineering*, ASE’13, pages 301–311, Nov. 2013. 344
644. O. Gurevich, M. A. Johnson, and A. E. Goldberg. Incidental verbatim memory for language. *Language and Cognition*, 2(1):45–78, May 2010. 169
645. R. K. Guy. The strong law of small numbers. *American Mathematical Monthly*, 95(8):697–712, Oct. 1988. 233
646. Hackerone. The 2018 hacker report. Technical report, hackerone, Dec. 2017. 62
647. J. Haidt. *The Righteous Mind*. Vintage books, 2012. 39
648. A. G. Haldane and R. Davies. The short long. Speech, May 2011. 29th Société Universitaire Européene de Recherches Financières Colloquium. 97
649. A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, and B. Baudry. Test them all, is it worth it? A ground truth comparison of configuration sampling strategies. In *eprint arXiv:cs.SE/1710.07980*, Oct. 2017. 130, 157
650. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov. 2012. 358
651. M. H. Halstead. A theoretical relationship between mental work and machine language programming. Technical Report CSD TR 67, Purdue University, Feb. 1972. 178
652. M. H. Halstead and P. M. Zislis. Experimental verification of two theorems of software physics. Technical Report TR 97, Purdue University, June 1973. 178
653. D. Z. Hambrick, F. L. Oswald, E. M. Altmann, E. J. Meinz, F. Gobet, and G. Campitelli. Deliberate practice: Is that all it takes to become an expert? *Intelligence*, 45(1):34–45, July-Aug. 2014. 35
654. M. Hamill and K. Goseva-Popstojanova. Exploring the missing link: an empirical study of software fixes. *Software Testing, Verification and Reliability*, 24(8):684–705, Dec. 2014. 201, 202
655. J. E. Hannay, D. I. K. Sjøberg, and T. Dybå. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, Feb. 2007. 6
656. M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. Report No. UCB/CSD-95-887, Computer Science Division, University of California Berkeley, Nov. 1995. 101
657. B. R. Harmon and N. I. Om. Schedule assessment methods for ballistic missile defense ground-based software development. IDA Paper P-3600, Institute for Defense Analyses, Aug. 2003. 115
658. J. Haskel and S. Westlake. *Capitalism without Capital: The Rise of the Intangible Economy*. Princeton University Press, 2018. 4, 55
659. H. Hata, C. Treude, R. G. Kula, and T. Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *eprint arXiv:cs.SE/1901.07440*, Jan. 2019. 130
660. L. Hatton. *Safer C : Developing Software for High-integrity and Safety-critical Systems*. McGraw-Hill, 1995. 166
661. L. Hatton. Reexamining the fault density-component size connection. *IEEE Software*, 14(2):89–97, Mar. 1997. 207
662. L. Hatton. How accurately do engineers predict software maintenance tasks? *Computer*, 40(2):64–69, Feb. 2007. 198, 328
663. R. Hatzinger and R. Dittrich. prefmod: An R package for modeling preferences based on paired comparisons, rankings, or ratings. *Journal of Statistical Software*, 48(10):1–31, May 2012. 334
664. M. D. Hauser, S. Carey, and L. B. Hauser. Spontaneous number representation in semi-free-ranging rhesus monkeys. *Proceedings of the Royal Society B: Biological Sciences*, 267(1445):829–833, Apr. 2000. 16
665. J. P. Haverty and R. L. Patrick. Programming languages and standardization in command and control. Research Memorandum RM-3447-PR, The RAND Corporation, Jan. 1963. 102
666. D. M. Hawkins. *Identification of Outliers*. Springer, 1980. 359
667. G. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. Context effects in multi-alternative decision making: Empirical data and a bayesian model. *Cognitive Science*, 36(3):498–516, Apr. 2012. 53
668. G. E. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. An optimal adjustment procedure to minimize experiment time in decisions with multiple alternatives. *Psychonomic Bulletin & Review*, 19(2):339–348, Apr. 2012. 342
669. J. A. Hawkins. *Efficiency and Complexity in Grammars*. Oxford University Press, 2007. 169
670. B. Hayes. Third base. *American Scientist*, 89(6):490–494, 2001. 5
671. S. Hazelhurst. Truth in advertising: Reporting performance of computer programs, algorithms and the impact of architecture and systems environment. *South African Computer Journal*, 46:24–37, Dec. 2010. 295
672. S. Head and J. Nelson. Data rights valuation in software acquisitions. Technical Report DRM-2012-001825-Final, CNA Analysis & Solutions, Sept. 2012. 64
673. A. Heathcote, S. Brown, and D. J. K. Mewhort. The power law repealed: The case for an exponential law of practice. *Psychonomic Bulletin & Review*, 7(2):185–207, Apr. 2000. 32
674. R. Heeks. The uneven profile of Indian software exports. Working Paper No. 3, University of Manchester, Oct. 1998. 56
675. J. Heer and M. Bostock. Crowdsourcing graphical perception: Using Mechanical Turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 2010, pages 203–212, Apr. 2010. 203

676. D. H. Helmer, S. Mackay, K. Selvey-Clinton, R. Yoon, and H. Fukukawa. Worldwide capital and fixed assets guide 2016. Technical Report EYG no. DL1528, EYGM Limited, 2016. 75
677. D. R. Helsel. *Statistics for Censored Environmental Data using Minitab and R*. John Wiley & Sons, second edition, 2012. 314
678. A. Hemel and R. Koschke. Reverse engineering variability in source code using clone detection-A case study for Linux variants of consumer electronic devices. In *19th Working Conference on Reverse Engineering*, WCRE'12, pages 357–366, Oct. 2012. 87
679. M. Hendrickson. 2010 state of the computer book market, Feb. 2011. <http://radar.oreilly.com/2011/02/2010-book-market-1.html>. 103
680. A. Henik and J. Tzelgov. Is three greater than five: The relation between physical and semantic size in comparison tasks. *Memory & Cognition*, 10(4):389–395, 1982. 46
681. J. Henrich. How adaptive cultural processes can produce maladaptive losses—The Tasmanian case. *American Antiquity*, 69(2):197–214, Apr. 2004. 69
682. J. Henrich. The evolution of costly displays, cooperation and religion: credibility enhancing displays and their implications for cultural evolution. *Evolution and Human Behavior*, 30(4):244–260, July 2009. 67
683. J. Henrich, M. Chudek, and R. Boyd. The big man mechanism: how prestige fosters cooperation and creates prosocial leaders. *Philosophical Transactions of The Royal Society B*, 370(1683):???, Dec. 2015. 109
684. J. Henrich and F. J. Gil-White. The evolution of prestige Freely conferred deference as a mechanism for enhancing the benefits of cultural transmission. *Evolution and Human Behavior*, 22(3):165–196, May 2001. 68
685. J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? Working Paper No. 139, German Data Forum (RatSWD), Apr. 2010. 17
686. J. Henrich and R. McElreath. Are peasants risk-averse decision makers? *Current Anthropology*, 43(1):172–181, Feb. 2002. 48
687. F. Hermans and E. Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 7–16, May 2015. 164
688. T. Herr, B. Schneier, and C. Morris. Taking stock: Estimating vulnerability rediscovery. Paper, Belfer Center for Science and International Affairs, Harvard Kennedy School, July 2017. 149
689. I. Herreraiz, G. Robles, J. M. González-Barahona, A. Capiluppi, and J. F. Ramíl. Comparison between SLOCs and number of files as size metrics for software evolution analysis. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, CSMR 2006, pages 206–213, Mar. 2006. 165
690. E. Herrmann, J. Call, M. Victoria, Hernández-Lloreda, B. Hare, and M. Tomasello. Humans have evolved specialized skills of social cognition: The cultural intelligence hypothesis. *Science*, 317(5843):1360–1366, Sept. 2007. 47, 68
691. R. Hersh. *18 Unconventional Essays on the Nature of Mathematics*. Springer, 2006. 138
692. K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE'13, pages 392–401, May 2013. 139, 359
693. K. Herzig and A. Zeller. Untangling changes. Submitted to MSR 2013, 2013. 359
694. T. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. In *eprint arXiv:stat.OT/1411.5279*, Nov. 2014. 246
695. M. Hicks, C. O'Malley, S. Nichols, and B. Anderson. Comparison of 2D and 3D representations for visualising telecommunication usage. *Behaviour & Information Technology*, 22(3):185–201, May 2003. 203
696. E. T. Higgins. Value from regulatory fit. *Current Directions in Psychological Science*, 14(4):209–213, 2005. 20
697. N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996. 137
698. M. Hilbert and P. López. Supporting online material for: The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, Apr. 2011. 84
699. J. Hill, L. C. Thomas, and D. E. Allen. Experts' estimates of task durations in software development projects. *International Journal of Project Management*, 18(1):13–21, Feb. 2000. 116
700. T. P. Hill. An evolutionary theory for the variability hypothesis. In *eprint arXiv:q-bio.PE/1703.04184*, Aug. 2018. 17
701. T. T. Hills, P. M. Todd, and M. N. Jones. Foraging in semantic fields: How we search through memory. *Topics in Cognitive Science*, 7(3):513–534, July 2015. 30
702. D. J. Hilton. The social context of reasoning: Conversational inference and rational judgment. *Psychological Bulletin*, 118(2):248–271, 1995. 40
703. A. Hindle, M. W. Godfrey, and R. C. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In *The 16th IEEE International Conference on Program Comprehension*, ICPC 2008, pages 133–142, June 2008. 307
704. T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K. ichi Matsumoto. The impact of a low level of agreement among reviewers in a code review process. In *IFIP International Conference on Open Source Systems*, OSS 2016, pages 97–110, May-June 2016. 159
705. S. C. Hirtle and J. Jonides. Evidence for hierarchies in cognitive maps. *Memory & Cognition*, 13(3):208–217, 1985. 46
706. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, Oct. 1969. 138
707. M. Hocko and T. Kalibera. Reducing performance non-determinism via cache-aware page allocation strategies. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW'10, pages 223–234, Jan. 2010. 351
708. A. Höfer. Exploratory comparison of expert and novice pair programmers. *Computing and Informatics*, 29(1):73–91, 2010. 282
709. D. D. Hoffman. *Visual Intelligence: How We Create What We See*. W. W. Norton, 2000. 15, 23
710. R. Hofman. *Behavioral Products Quality Assessment Model on the Software Market*. PhD thesis, Poznan University of Economics, Oct. 2011. 131
711. J. Hofmeister, J. Siegmund, and D. V. Holt. Shorter identifier names take longer to comprehend. In *24th International Conference on Software Analysis, Evolution and Reengineering*, SANER 2017, pages 217–227, Feb. 2017. 177
712. G. Hofstede. *Culture's Consequences: International Differences in Work-Related Values*. Sage Publications, abridged edition, 1984. 67
713. R. M. Hogarth and H. J. Einhorn. Order effects in belief updating: The belief-adjustment model. *Cognitive Psychology*, 24(1):1–55, Jan. 1992. 34, 35
714. R. M. Hogarth, C. R. M. McKenzie, B. J. Gibbs, and M. A. Marquis. Learning from feedback: Exactness and incentives. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 17(4):734–752, 1991. 35
715. B. D. Holbrook and W. S. Brown. A history of computing research at Bell Laboratories (1937–1975). Computing Science Technical Report No. 99, AT&T Bell Laboratories, 1982. 82
716. M. Holdway. An alternative methodology: Valuing quality change for microprocessors in the PPI. In *Issues in Measuring Price Change and Consumption*. Bureau of Labor Statistics, June 2000. 4
717. W. B. Holland. Soviet cybernetics technology: viii. Report on the algorithmic language ALGEC (final version). Research Memorandum RM-5136-PR, The RAND Corporation, Dec. 1966. 96
718. J. K. Hollmann. Estimate accuracy: Dealing with reality. *Cost Engineering Journal*, 54(6):17–27, Nov.-Dec. 2012. 113
719. A. A. Hook, B. Brykczyński, C. W. McDonald, S. H. Nash, and C. Youngblut. A survey of computer programming languages currently used in the Department of Defense. IDA PAPER P-3054, Institute for Defense Analyses, Jan. 1995. 103
720. R. Hoosain. Correlation between pronunciation speed and digit span size. *Perception and Motor Skills*, 55:1128–1128, 1982. 342
721. R. Hoosain and F. Salili. Language differences, working memory, and mathematical ability. In M. M. Grunberg, P. E. Morris, and R. N. Sykes, editors, *Practical aspects of memory: Current research and issues*, volume 2, pages 512–517. John Wiley & Sons, Inc, 1988. 27

722. M. Hoppe and S. Hanenberg. Do developers benefit from generic types? An empirical comparison of generic and raw types in Java. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, OOPSLA'13, pages 457–474, Oct. 2013. [168](#)
723. W. Hoppitt and K. N. Laland. *Social Learning: An Introduction to Mechanisms, Methods, and Models*. Princeton University Press, July 2013. [67](#)
724. W. Hordijk, M. L. Poncino, and R. Wieringa. Harmfulness of code duplication a structured review of the evidence. In *13<sup>th</sup> International Conference on Evaluation and Assessment in Software Engineering*, EASE'09, pages 88–97, Apr. 2009. [118](#)
725. M. R. Horton. *Portable C Software*. Prentice-Hall, Inc, Upper Saddle River, NJ 07458, USA, 1990. [166](#)
726. A. D. Householder and J. M. Foote. Probability-based parameter selection for black-box fuzz testing. Technical Note CMU/SEI-2012-TN-019, Software Engineering Institute, Carnegie Mellon University, Aug. 2012. [146](#)
727. M. W. Howard and M. J. Kahana. Context variability and serial position effects in free recall. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 25(4):923–941, 1999. [30](#)
728. J. Howison and J. B. Herbsleb. Incentives and integration in scientific software production. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW'13, pages 459–470, Mar. 2013. [67, 85](#)
729. L. Hribar, S. Bogovac, and Z. Marinčić. Implementation of fault slip through in design phase of the project. In *miproBIS 2008: International Conference on Business Intelligence Systems*, page ???, May 2008. [158](#)
730. http archive. <https://httparchive.org>, July 2018. [88](#)
731. X. Huang, J. Xie, N. O. Otecko, and M. Peng. Accessibility and update status of published software: Benefits and missed opportunities. *Frontiers in Research Metrics and Analytics*, 2(??):???, Feb. 2017. [134](#)
732. B. A. Huberman. The dynamics of organizational learning. *Computational & Mathematical Organization Theory*, 7(2):145–153, Aug. 2001. [69](#)
733. H. Huijgens and R. van Solingen. Measuring best-in-class software releases. In *Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pages 137–146, Oct. 2013. [116](#)
734. H. Huijgens and F. Vogezelang. Do estimators learn? On the effect of a positively skewed distribution of effort data on software portfolio productivity. Technical Report TUD-SERG-2016-004, Delft University of Technology, 2016. [70](#)
735. J. C. Hull. *Options, Futures, and other Derivatives*. Pearson, seventh edition, Oct. 2010. [59](#)
736. C. Hulme, S. Maughan, and G. D. A. Brown. Memory for familiar and unfamiliar words: Evidence for a long-term memory contribution to short-term memory span. *Journal of Memory and Language*, 30(6):685–701, 1991. [28](#)
737. C. R. Hulten. Decoding Microsoft: Intangible capital as a source of company growth. Working Paper 15799, National Bureau of Economic Research, USA, Mar. 2010. [74](#)
738. R. Hundt, E. Raman, M. Thuresson, and N. Vachharajani. MAO—an extensible micro-architectural optimizer. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO'11, pages 1–10, Apr. 2011. [348](#)
739. S. Hunold and A. Carpen-Amarie. MPI benchmarking revisited: Experimental design and reproducibility. In *eprint arXiv:cs.DC/1505.07734v3*, Sept. 2015. [347](#)
740. S. Hunold, A. Carpen-Amarie, and J. L. Träff. Reproducible MPI micro-benchmarking isn't as easy as you think. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA'14, pages 69–76, Sept. 2014. [223, 224](#)
741. E. Hunt. The Whorfian hypothesis: A cognitive psychology perspective. *Psychological Review*, 98(3):377–389, 1991. [180](#)
742. J. E. Hunter, F. L. Schmidt, and M. K. Judesch. Individual differences in output variability as a function of job complexity. *Journal of Applied Psychology*, 75(1):28–42, Feb. 1990. [52](#)
743. M. J. Hurlstone, G. J. Hitch, and A. D. Baddeley. Memory for serial order across domains: An overview of the literature and directions for future research. *Psychonomic Bulletin & Review*, 140(2):229–373, Mar. 2014. [30](#)
744. A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 111–122, Mar. 2012. [156](#)
745. IBM. Specifications for the IBM mathematical FORmula TRANSlating system, FORTRAN. Programming Research Group, Applied Science Division, International Business Machines Corporation, Nov. 1954. [102](#)
746. R. Jerusalimsky, L. H. de Figueiredo, and W. Celes. The evolution of Lua. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 1–26, June 2007. [82](#)
747. J. Iivonen. Identifying and characterizing highly performing testers—A case study in three software product companies. Thesis (m.s.), Helsinki University of Technology, Department of Computer Science and Engineering, Oct. 2009. [52](#)
748. I. Imbo and J.-A. LeFevre. Cultural differences in complex addition: Efficient Chinese versus adaptive Belgians and Canadians. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 35(6):1465–1476, Nov. 2009. [44](#)
749. I. Imbo, A. Vandierendonck, and E. Vergauwe. The role of working memory in carrying and borrowing. *Psychological Research*, 71(4):467–483, July 2007. [44](#)
750. L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE'14, pages 435–445, June 2014. [161](#)
751. Intel. 6th generation Intel processor family. Specification update 332689-010EN, Intel Corporation, Apr. 2017. [151](#)
752. International telecommunication union. website, July 2012. <http://www.itu.int>. [152](#)
753. J. P. A. Ioannidis. Contradicted and initially stronger effects in highly cited clinical research. *JAMA*, 294(2):218–228, July 2005. [10](#)
754. J. P. A. Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):e124, Aug. 2005. [245](#)
755. A. Iosup, M. Jan, O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. Rapport de recherche no 6172, Institut National de Recherche en Informatique et en Automatique, Apr. 2007. [157](#)
756. G. Irlam. Unix file size survey-1993. <http://www.base.com/gordoni/ufs93.html>, Sept. 1993. [225](#)
757. F. Irving. Github users since service started. [https://classic.scrapewiki.com/scrapers/github\\_users\\_each\\_year/](https://classic.scrapewiki.com/scrapers/github_users_each_year/), Mar. 2016. [78](#)
758. ISO. *ISO/IEC Guide 25:1990 General requirements for the competence of calibration and testing laboratories*. International Organization for Standardization, 1990. [160](#)
759. ISO. *ISO/IEC 9945:2008 Information technology – Portable Operating System Interface (POSIX®)*. ISO, 2008. [104](#)
760. ISO SC22. *ISO/IEC 18009:1999 Information technology – Programming languages – Ada: Conformity assessment of a language processor*. International Organization for Standardization, 1990. Last reviewed and confirmed in 2015. [160](#)
761. ISO SC22. *ISO/IEC 13210:1999 Information technology – Requirements and guidelines for test methods specifications and test method implementation for measuring conformance to POSIX standards*. International Organization for Standardization, 1999. [160](#)
762. A. Israeli and D. G. Feitelson. The Linux kernel as a case study in software evolution. *Journal of Systems and Software*, 83(3):485–501, Mar. 2010. [178, 268, 269, 275, 296](#)
763. R. K. Iyer, S. E. Butner, and E. J. McCluskey. An exponential failure/load relationship: Results of a multi-computer statistical study. Technical Report CRC-81-6, Computer Systems Laboratory, Stanford University, Aug. 1981. [140](#)
764. M. Y. Jaber. Learning and forgetting models and their applications. In A. B. Badiru, editor, *Handbook of Industrial and Systems Engineering*, chapter 30. CRC Press-Taylor & Francis Group, Dec. 2005. [69](#)

765. A. N. Jackson. Formats over time: Exploring UK web history. In *eprint arXiv:cs.DL/1210.1714v1*, Oct. 2012. 100
766. J. Jacobs and B. Rudis. *Data-Driven Security: Analysis, Visualization and Dashboards*. John Wiley & Sons, Inc, 2014. 191, 365
767. R. Jaeschke. *Portability and the C Language*. Hayden Books, 4300 West 62nd Street, Indianapolis, IN 46268, USA, 1989. 166
768. L. R. Jager and J. T. Leek. Empirical estimates suggest most published research is true. *Biostatistics*, 15(1):1–12, 2014. 245
769. B. Jamtveit, E. Jettestuen, and J. Mathiesen. Scaling properties of European research units. *PNAS*, 106(32):13160–13163, Aug. 2009. 98
770. A. R. Jansen. *Encoding and Parsing of Algebraic Expressions by Experienced Users of Mathematics*. PhD thesis, School of Computer Science and Software Engineering, Monash University, Jan. 2002. 25
771. A. R. Jansen, K. Marriott, and G. W. Yelland. Parsing of algebraic expressions by experienced users of mathematics. *European Journal of Cognitive Psychology*, 19(2):286–320, 2007. 25
772. C. J. M. Jansen and M. M. W. Pollmann. On round numbers: Pragmatic aspects of numerical expressions. *Journal of Quantitative Linguistics*, 8(3):187–201, 2001. 45
773. J. J. Jenkins. Remember that old theory of memory? Well, forget it! *American Psychologist*, 29(11):785–795, 1974. 169
774. J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering*, 22(1):547–578, Feb. 2017. 86
775. Y. Jiang, B. Adams, and D. M. German. Will my patch make it? And how fast? Case study on the Linux kernel. In *10th Working Conference on Mining Software Repositories*, MSR’13, pages 101–110, May 2013. 134, 135
776. D. D. P. Johnson, N. B. Weidmann, and L.-E. Cederman. Fortune favours the bold: An agent-based model reveals adaptive advantages of overconfidence in war. *PLoS ONE*, 6(6):e20851, Apr. 2011. 50
777. J. A. Johnson. Measuring thirty facets of the Five Factor model with a 120-item public domain inventory: Development of the IPIP-NEO-120. *Journal of Research in Personality*, 51:78–89, Aug. 2014. 47
778. L. Johnson. Applied data research inc. (ADR). Technical report, Computer History Museum, Feb. 2010. 97
779. P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Empirical Software Engineering*, 4(4):317–349, Dec. 1999. 358
780. W. L. Johnson. *Intention-Based Diagnosis of Novice Programming Errors*. Morgan Kaufmann Publishers, Inc, 1986. 179
781. C. I. Jones. The facts of economic growth. In J. B. Taylor and H. Uhlig, editors, *Handbook of Macroeconomics, Volume 2A*, chapter 1, pages 3–69. Elsevier B. V., Nov. 2016. 4
782. D. Jones. Why userspace sucks—or 101 really dumb things your app shouldn’t do. In *Proceedings of the Linux Symposium: Volume One*, pages 441–450, July 2006. 351
783. D. M. Jones. Who guards the guardians? [www.knosof.co.uk/whoguard.html](http://www.knosof.co.uk/whoguard.html), 1992. 162
784. D. M. Jones. The  $7 \pm 2$  urban legend. MISRA C 2002 conference <http://www.knosof.co.uk/cbook/misart.pdf>, Oct. 2002. 27, 341
785. D. M. Jones. The new C Standard: An economic and cultural commentary. Knowledge Software, Ltd, 2005. 7, 84, 104, 127, 152, 174, 176, 179, 183, 184, 185, 187, 193, 204, 229, 280, 342, 365
786. D. M. Jones. Developer beliefs about binary operator precedence. *C Vu*, 18(4):14–21, Aug. 2006. 33, 34, 48, 334, 342, 355, 356
787. D. M. Jones. Operand names influence operator precedence decisions. *C Vu*, 20(1):5–11, Feb. 2008. 48, 174, 355, 356
788. D. M. Jones. Deciding between if and switch when writing code. *C Vu*, 21(5):14–20, Nov. 2009. 184
789. D. M. Jones. Developer characterization of data structure fields decisions. *C Vu*, 20(6):14–18, Jan. 2009. 39, 229, 230, 333, 334, 355, 356
790. D. M. Jones. Birth month of compiler writers. The Shape of Code: blog, Feb. 2012. <http://shape-of-code.coding-guidelines.com>. 325
791. D. M. Jones. Effects of risk attitude on recall of assignment statements. *C Vu*, 23(6):19–22, Jan. 2012. 48
792. D. M. Jones. Code & data used in: Evidence-based software engineering: Based on the publicly available data. <http://www.github.com/Derek-Jones/ESEUR>, 2019. 2
793. M. N. Jones and D. J. K. Mewhort. Case-sensitive letter and bigram frequency counts from large-scale English corpora. *Behavior Research Methods, Instruments, & Computers*, 36(3):388–396, 2004. 178
794. R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, Oct. 2017. 330
795. M. R. Jongerden. *Model-based energy analysis of battery powered systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, Dec. 2010. 348
796. M. Jørgensen. An empirical study of software maintenance tasks. *Software Maintenance: Research and Practice*, 7(1):27–48, Jan. 1995. 282
797. M. Jørgensen. Regression models of software development effort estimation accuracy and bias. *Empirical Software Engineering*, 9(4):297–394, Apr. 2004. 113, 117
798. M. Jørgensen. Better selection of software providers through trial-sourcing. *IEEE Software*, 33(5):48–53, Sept.–Oct. 2016. 116, 118
799. M. Jørgensen. A survey on the characteristics of projects with success in delivering client benefits. *Information and Software Technology*, 78:83–94, Oct. 2016. 126
800. M. Jørgensen and G. J. Carelius. An empirical study of software project bidding. *IEEE Transactions on Software Engineering*, 30(12):953–969, Dec. 2004. 119, 252
801. M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, 116:133–145, June 2016. 245
802. M. Jørgensen and S. Grimstad. Software development estimation biases: The role of interdependence. *IEEE Transactions on Software Engineering*, 38(3):677–693, May 2012. 17, 21
803. M. Jørgensen, U. Indahl, and D. I. K. Sjøberg. Software effort estimation by analogy and “regression toward the mean”. *Journal of Systems and Software*, 68(3):253–262, Dec. 2003. 267
804. M. Jørgensen and K. Moløkken. Eliminating over-confidence in software development effort estimates. In F. Bomarius and H. Iida, editors, *Product Focused Software Process Improvement*, volume 3009 of *Lecture Notes in Computer Science*, pages 174–184. Springer Berlin Heidelberg, Apr. 2004. 253
805. M. Jørgensen and K. Moløkken. Understanding reasons for errors in software effort estimates. *IEEE Transactions on Software Engineering*, 30(12):993–1007, Dec. 2004. 19
806. M. Jørgensen and K. Moløkken. How large are software cost overruns? A review of the 1994 CHAOS report. *Journal Information and Software Technology*, 48(4):297–301, Apr. 2006. 112
807. M. Jørgensen and D. I. K. Sjøberg. The impact of customer expectation on software development effort estimates. *International Journal of Project Management*, 22(4):317–325, May 2004. 20, 113
808. M. Jørgensen and D. I. K. Sjøberg. Learning from experience in a software maintenance environment. *Journal of Computer Science*, 1(4):538–542, Apr. 2005. 275
809. D. Joseph, W. F. Boh, S. Ang, and S. A. Slaughter. The career paths less (or more) travelled: A sequence analysis of IT career histories, mobility patterns, and career success. *MIS Quarterly*, 36(2):427–452, June 2012. 65
810. S. Kahrs. Mistakes and ambiguities in the definition of standard ML. LFCS report ECS-LFCS-93-257, University of Edinburgh, Scotland, Apr. 1993. 139
811. J. W. Kalat. *Biological Psychology*. Wadsworth, seventh edition, 2001. 16
812. T. Kalibera, L. Bulej, and P. Tůma. Benchmark precision and random initial state. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SPECTS 2005, pages 853–862. Society for Modeling and Simulation (SCS), July 2005. 351
813. A. Kaltenbrunner, V. Gómez, A. Moghnieh, R. Meza, J. Blat, and V. López. Homogeneous temporal activity patterns in a large online communication space. In *eprint arXiv:cs.NI/0708.1579*, Aug. 2007. 224

814. C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel. Distance-based sampling of software configuration spaces. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE'19, pages 1084–1094, May 2019. 154
815. D. Kaminsky, M. Eddington, and A. Cecchetti. Showing how security has (and hasn't) improved, after ten years of trying. CanSecWest Applied Security Conference, Dec. 2011. 148, 149, 150
816. T. Kamiya. How code skips over revisions. In *Proceedings of the 5th International Workshop on Software Clones*, IWSC 2011, pages 69–70, May 2011. 180
817. V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, Apr. 2007. 6
818. P. Kampstra and C. Verhoef. Reliability of function point counts. ???, ???(??):???, Apr. 200? 116
819. P. Kampstra and C. Verhoef. Benchmarking the expected loss of a federal IT portfolio. ???, July 2009. 263
820. T. Kanda, T. Ishio, and K. Inoue. Approximating the evolution history of software from source code. *IEICE Transactions on Information and Systems*, E98-D(6):1185–1193, June 2015. 332
821. C. Kaner. Liability for defective documentation. In *Proceedings of the 21st annual international conference on Documentation*, SIGDOC'03, pages 192–197, Oct. 2003. 130, 155
822. C. Kaner and D. Pels. *Bad Software: What To Do When Software Fails*. John Wiley & Sons, Inc, 1998. 143
823. Y. Kang, B. Ray, and S. Jana. APEx: Automated inference of error specifications for C APIs. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 472–482, Sept. 2016. 161
824. S. J. Karau and K. D. Williams. Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychology*, 65(4):681–706, Oct. 1993. 72
825. D. Karlis and E. Xekalaki. Mixed poisson distributions. *International Statistical Review*, 73(1):35–58, Apr. 2005. 217
826. J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, Sept. 1997. 128
827. D. S. Katz, K. McHenry, C. Reinking, and R. Haines. Research software development & management in universities: Case studies from Manchester's RSDS group, Illinois' NCSA, and Notre Dame's CRC. In *eprint arXiv:cs.SE/1903.00732*, Mar. 2019. 98
828. D. Kawrykow and M. P. Robillard. Non-essential changes in version histories. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE'11, pages 351–360, May 2011. 131
829. M. Kazandjieva, B. Heller, O. Gnawali, P. Levis, and C. Kozyrakis. Green enterprise computing data: Assumptions and realities. In *Proceedings of the 2012 International Green Computing Conference*, IGCC'12, pages 1–10, June 2012. 84
830. M. Keil and D. Robey. Blowing the whistle on troubled software projects. *Communications of the ACM*, 44(4):87–93, Apr. 2001. 121
831. P. Keil, J. M. Bennett, B. Bourgeois, G. E. G.-P. na, A. A. M. MacDonald, C. Meyer, K. S. Ramirez, and B. Yguel. From computer operating systems to biodiversity: co-emergence of ecological and evolutionary patterns. *PNAS*, ???(??):???, Aug. 2016. 86, 87
832. C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987. 115
833. Z. Kenessey. The primary, secondary, tertiary and quaternary sectors of the economy. *The Review of Income and Wealth*, 33(4):359–385, Dec. 1987. 55
834. D. O. Kennedy and A. B. Scholey. Glucose administration, heart rate and cognitive performance: effects of increasing mental effort. *Psychopharmacology*, 149(1):63–71, May 2000. 52
835. B. W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley, 1999. 166
836. M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of COLING-90*, pages 205–210, 1990. 176
837. E. Keuleers, P. Lacey, K. Rastle, and M. Brysbaert. The British lexicon project: Lexical decision data for 28,730 monosyllabic and disyllabic English words. *Behavior and Research Methods*, 44(1):287–304, Mar. 2012. 31
838. H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan. Prioritizing the devices to test your app on: A case study of Android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 610–620, Nov. 2014. 75
839. L. M. Khan. Amazon's antitrust paradox. *The Yale Law Journal*, 126(3):710–805, Jan. 2017. 91
840. P.-V. Khuong and P. Morin. Array layouts for comparison-based searching. In *eprint arXiv:cs.DS/1509.05053*, Mar. 2017. 350, 351
841. P. D. Killworth and H. R. Bernard. Informant accuracy in social network data. *Human Organization*, 35(3):269–286, Sept.–Dec. 1976. 338
842. D. Kim, E. Murphy-Hill, C. Parnin, C. Bird, and R. Garcia. The reaction of open-source projects to new language features: An empirical study of C# generics. *The Journal of Object Technology*, 12(4):1–26, Nov. 2013. 181
843. H. Kim. *Informed Storage Management for Mobile Platforms*. PhD thesis, College of Computing, Georgia Institute of Technology, Dec. 2012. 350
844. S. Kim. The classification of information and communication technology investment in financial accounting. Thesis (m.s.), School of Information Technologies, University of Sydney, 2013. 75
845. K. Kina, M. Tsunoda, H. Hata, H. Tamada, and H. Igaki. Analyzing the decision criteria of software developers based on prospect theory. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER'16, pages 644–648, Mar. 2016. 50
846. D. King and C. Janiszewski. The sources and consequences of the fluent processing of numbers. *Journal of Marketing Research*, XLVIII(2):327–341, 2011. 45
847. J. King and M. A. Just. Individual differences in syntactic processing: The role of working memory. *Journal of Memory and Language*, 30:580–602, 1991. 28
848. W. Kintsch. *Comprehension: A paradigm for cognition*. Cambridge University Press, 1998. 172
849. W. Kintsch and J. Keenan. Reading rate and retention as a function of the number of propositions in the base structure of sentences. *Cognitive Psychology*, 5(3):257–274, Nov. 1973. 169
850. W. Kintsch, E. Kozminsky, W. J. Streby, G. McKoon, and J. M. Keenan. Comprehension and recall of text as a function of content variables. *Journal of Verbal Learning and Verbal Behavior*, 14(2):196–214, Apr. 1975. 169
851. W. Kintsch, T. S. Mandel, and E. Kozminsky. Summarizing scrambled stories. *Memory & Cognition*, 5(5):547–552, 1977. 172
852. K. N. Kirby and R. J. Herrnstein. Preference reversals due to myopic discounting of delayed reward. *Psychological Science*, 6(2):83–89, Mar. 1995. 51
853. D. Kirsh and P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, Oct. 1994. 18
854. L. B. Kish. Moore's law and the energy requirement of computing versus performance. *IEE Proceedings-Circuits, Devices and Systems*, 151(2):190–194, Apr. 2004. 155
855. J. V. Kistowski, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev. Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE'15, pages 223–234, Jan. 2015. 233
856. S. Kitayama and M. Karasawa. Implicit self-esteem in Japan: Nameletters and birthday numbers. *Personality & Social Psychology Bulletin*, 23(7):736–742, 1997. 95
857. B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *The Journal of Systems and Software*, 64(1):57–77, Oct. 2002. 113, 117
858. B. A. Kitchenham and N. R. Taylor. Software project development cost estimation. *The Journal of Systems and Software*, 5(4):267–278, Nov. 1985. 126
859. A. Kivi, T. Smura, and J. Töyli. Technology product evolution and the diffusion of new product features. *Technological Forecasting & Social Change*, 79(1):107–126, Jan. 2012. 77

860. D. Klahr, W. G. Chase, and E. A. Lovelace. Structure and process in alphabetic retrieval. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 9(3):462–477, 1983. 30
861. J. Klayman and Y.-W. Ha. Confirmation, disconfirmation, and information in hypothesis testing. *Psychological Review*, 94(2):211–228, 1987. 21
862. B. Klein. The decision making problem in development. In Universities-National Bureau Committee for Economic Research, Committee on Economic Growth of the Social Science Research Council, editor, *The Rate and Direction of Inventive Activity: Economic and Social Factors*, chapter 19, pages 477–508. Princeton University Press, 1962. 121
863. S. B. Klein, L. Cosmides, J. Tooby, and S. Chance. Decisions and the evolution of memory: Multiple systems, multiple functions. *Psychological Review*, 109(2):306–329, 2002. 25
864. J. Kleinberg and M. Raghu. Team performance with test scores. In *eprint arXiv:cs.DS/1506.00147v2*, Mar. 2018. 131
865. S. Kleinschmager, R. Robbes, A. Stefk, S. Hanenberg, and É. Tanner. Do static type systems improve the maintainability of software systems? An empirical study. In *20th International Conference on Program Comprehension*, ICPC’12, pages 153–162, June 2012. 168
866. S. Klepper and K. L. Simons. Technological extinctions of industrial firms: An inquiry into their nature and causes. *Industrial and Corporate Change*, 6(2):379–460, Mar. 1997. 97
867. P. Klint, D. Landman, and J. Vinju. Exploring the limits of domain model recovery. In *29th IEEE International Conference on Software Maintenance*, ICSM’13, pages 120–129, Sept. 2013. 117
868. K. E. Knight. Changes in computer performance. *Datamation*, 12(9):40–54, Sept. 1966. 345
869. K. E. Knight. Evolving computer performance 1963–1967. *Datamation*, 14(1):31–35, Jan. 1968. 5, 345
870. D. E. Knuth. The errors of TeX. *Software—Practice and Experience*, 19(7):607–685, 1989. 141
871. D. Kobak, S. Shpilkin, and M. S. Pshenichnikov. Integer percentages as electoral falsification fingerprints. In *eprint arXiv:stat.AP/1410.6059v4*, June 2016. 366
872. A. Koenig. *C Traps and Pitfalls*. Addison-Wesley, 1989. 166
873. R. Kohavi and R. Longbotham. Unexpected results in online controlled experiments. *ACM SIGKDD Explorations Newsletter*, 12(2):31–35, Dec. 2010. 342
874. P. A. Kolers. Reading A year later. *Journal of Experimental Psychology: Human Learning and Memory*, 2(3):554–565, 1976. 174
875. P. A. Kolers and D. N. Perkins. Spatial and ordinal components of form perception and literacy. *Cognitive Psychology*, 7(2):228–267, Apr. 1975. 173
876. J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54, July–Sept. 2011. 3
877. A. Koriat. How do we know that we know? The accessibility model of the feeling of knowing. *Psychological Review*, 100(4):609–639, 1993. 29
878. T. D. Korson and V. K. Vaishnavi. An empirical study of the effects of modularity on program modifiability. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, chapter 12, pages 168–186. Ablex Publishing Corporation, June 1986. 166
879. A. G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew. Theory of relative defect proneness: Replicated studies on the functional form of the size-defect relationship. *Empirical Software Engineering*, 13(5):473–498, Oct. 2008. 154
880. J. Kossik. Clark’s sector model for US economy 1850–2009. website, 2011. <http://www.63alfred.com/whomakesit/clarksmodel.htm>. 55
881. S. M. Kosslyn. *Graph Design for the Eye and Mind*. Oxford University Press, 2006. 206
882. S. M. Kosslyn and S. P. Shwartz. Empirical constraints on theories of visual imagery. In J. Long and A. D. Baddeley, editors, *Attention and Performance IX*, pages 241–260. Lawrence Erlbaum Associates, 1981. 18
883. KPMG. Project Tesla due diligence assistance. submitted as evidence in court case, Aug. 2011. 74
884. E. Krevat, J. Tucek, and G. R. Ganger. Disks are like snowflakes: No two are alike. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS’13, May 2013. 350
885. G. Kroah-Hartman. Linux kernel statistics. website, June 2016. <https://www.github.com/gregkh/kernel-history>. 265, 266, 291
886. J. K. Kruschke. Human category learning: Implications for backpropagation models. *Connection Science*, 5(1):3–36, 1993. 32, 33
887. J. K. Kruschke. Dimensional relevance shifts in category learning. *Connection Science*, 8(2):225–247, June 1996. 32, 33
888. E. C. Kubie. Recollections of the first software company. *IEEE Annals of the History of Computing*, 16(2):65–71, June 1994. 97
889. M. Kubovy and M. van den Berg. The whole is equal to the sum of its parts: A probabilistic model of grouping by proximity and similarity in regular patterns. *Psychological Review*, 115(1):131–154, 2008. 23, 24
890. B. M. Kuhn, Free Software Foundation, Inc., Software Freedom Law Center, A. K. Sebro, Jr., D. Gingerich, and C. Legal. *Copyleft and the GNU General Public License: A Comprehensive Tutorial and Guide*. [copyleft.org](http://copyleft.org), 2018. 62
891. D. R. Kuhn and D. R. Wallace. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, June 2004. 160
892. M. Kuhrmann, C. Konopka, P. Nelleman, P. Diebold, and J. Münch. Software process improvement: Where is the evidence? In *Proceedings of the 2015 International Conference on Software and System Process*, pages 107–116, Aug. 2015. 6
893. R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. In *eprint arXiv:cs.SE/1709.04621*, Sept. 2017. 10
894. R. Kumar. The business of scaling. *IEEE Solid-State Circuits Society Newsletter*, 12(1):22–26, 2007. 4
895. S. Kumar. Enforcing the GNU GPL. *Journal of Law, Technology & Policy*, 2006(1), 2006. 64
896. G. Kunda. *Engineering Culture: Control and Commitment in a High-Tech Corporation*. Temple University Press, 1992. 67
897. P. Küngas, S. Vakulenko, M. Dumas, C. Parra, and F. Casati. Reverse-engineering conference rankings: What does it take to make a reputable conference? *Scientometrics*, 96(2):651–665, Aug. 2013. 9
898. G. Kunst. Language popularity. <http://langpop.corger.nl/results>, 2013. 270
899. R. Kurzban, A. Duckworth, J. W. Kable, and J. Myers. An opportunity cost model of subjective effort and task performance. *Behavioral and Brain Sciences*, 36(6):661–679, Dec. 2013. 22
900. D. S. Kusumo, M. Staples, L. Zhu, and R. Jeffery. Analyzing differences in risk perceptions between developers and acquirers in OTS-based custom software projects using stakeholder analysis. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’12, pages 69–78, Sept. 2012. 121
901. T. Labiner. A big decision: Lease or buy? *Computers and Automation*, 6(10):6–8, Oct. 1957. 94
902. W. Labov. The boundaries of words and their meaning. In C.-J. N. Bailey and R. W. Shuy, editors, *New ways of analyzing variation of English*, pages 340–373. Georgetown Press, 1973. 38, 39
903. W. Labov. *Principles of Linguistic Change, volume 3: Cognitive and Cultural Factors*. Wiley-Blackwell, 2010. 174
904. J. C. Lagarias. *The Kepler Conjecture: The Hales-Ferguson Proof by Thomas C. Hales Samuel P. Ferguson*. Springer, 2010. 138
905. K. Laitinen. *Natural naming in software development and maintenance*. PhD thesis, University of Oulu, Finland, Oct. 1995. 175
906. G. Lakoff and M. Johnson. *Metaphors We Live By*. The University of Chicago Press, 1980. 42, 95
907. B. L. Lambert, K.-Y. Chang, and P. Gupta. Effects of frequency and similarity neighborhoods on pharmacists’ visual perception of drug names. *Social Science and Medicine*, 57(10):1939–1955, Nov. 2003. 176
908. R. Lämmel, E. Pek, and J. Starek. Large-scale, AST-based API-usage analysis of open-source Java projects. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC’11, pages 1317–1324, Mar. 2011. 184, 185

909. T. K. Landauer. How much do people remember? Some estimates of the quantity of learned information in long-term memory. *Cognitive Science*, 10:477–493, 1986. 52
910. R. M. Landers, J. B. Rebitzer, and L. J. Taylor. Rat race reduce: Adverse selection in the determination of work hours in law firms. *The American Economic Review*, 86(3):329–348, June 1996. 70
911. D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju. Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions. *Journal of Software: Evolution and Process*, 28(7):589–618, July 2016. 153, 165, 178
912. D. Landy, D. Brookes, and R. Smout. Abstract numeric relations and the visual structure of algebra. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 40(5):1404–1418, Sept. 2014. 23
913. D. Landy, A. Charlesworth, and E. Ottmar. Categories of large numbers in line estimation. *Cognitive Science*, 41(2):326–353, Mar. 2017. 44
914. D. Landy and R. L. Goldstone. Proximity and precedence in arithmetic. *The Quarterly Journal of Experimental Psychology*, 63(10):1953–1968, Oct. 2010. 192
915. D. Landy, B. Guay, and T. Marghetis. Bias and ignorance in demographic perception. *Psychonomic Bulletin & Review*, 25(5):1606–1618, Oct. 2018. 46
916. E. J. Langer. The illusion of control. *Journal of Personality and Social Psychology*, 32(2):311–328, 1975. 50
917. R. N. Langlois. External economies and economic progress: The case of the microcomputer industry. *The Business History Review*, 66(1):1–50, 1992. 83
918. LANL. Lanl failure data. <http://institute.lanl.gov/data/lanldata.shtml>, 2006. 157
919. L. Lapointe and S. Rivard. A multilevel model of resistance to information technology implementation. *MIS Quarterly*, 29(3):461–492, Sept. 2005. 110
920. I. Larkin. The cost of high-powered incentives: Employee gaming in enterprise software sales. Technical Report 13-073, Harvard Business School, Feb. 2013. 79
921. C. Larman and V. R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, June 2003. 122, 123
922. J. Larres. Performance variance evaluation on Mozilla Firefox. Thesis (m.s.), Victoria University of Wellington, May 2012. 352, 353
923. B. Latané and J. M. Darley. Bystander "apathy". *American Scientist*, 57(2):244–269, June–Sept. 1969. 72
924. B. Latané, K. Williams, and S. Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37(6):822–832, 1979. 72
925. R. Latorre. Effects of developer experience on learning and applying unit test-driven development. *IEEE Transactions on Software Engineering*, 40(4):381–395, Apr. 2014. 33, 34
926. P. R. Laughlin. *Group Problem Solving*. Princeton University Press, Apr. 2015. 72
927. E. Laukkanen, M. Paasivaara, J. Itkonen, C. Lassenius, and T. Arvonen. Towards continuous delivery by reducing the feature freeze period: A case study. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, ICSE-SEIP'17, pages 23–32, May 2017. 130
928. S. Laumer, C. Maier, A. Eckhardt, and T. Weitzel. Work routines as an object of resistance during information systems implementations: theoretical foundation and empirical evidence. *European Journal of Information Systems*, 25(4):317–343, July 2016. 110
929. L. Lauterbach. Development of N-version software samples for an experiment in software fault tolerance. NASA Contractor Report 178363, Software Research and Development Center for Digital Systems Research, Sept. 1987. 118, 151
930. E. Lazear and M. Gibbs. *Personnel Economics for Managers*. John Wiley & Sons, Inc, second edition, 2007. 98
931. C. Lebiere. *The Dynamics of Cognition: An ACT-R Model of Cognitive Arithmetic*. PhD thesis, Carnegie Mellon University, Nov. 1998. 44
932. P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):1–22, Aug. 2007. 155
933. A. L. Lederer and J. Prasad. Causes of inaccurate software development cost estimates. *Journal of Systems and Software*, 31(2):125–134, Nov. 1995. 113
934. B. C. Lee and D. M. Brooks. Regression modeling strategies for microarchitecture performance and power prediction. Technical Report TR-08-06, Division of Engineering and Applied Sciences, Harvard University, Mar. 2006. 302, 343
935. D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, HPCA, pages 489–501, Feb. 2015. 351
936. M. D. Lee, K. A. Gluck, and M. M. Walsh. Understanding the complexity of simple decisions: Modeling multiple behaviors and switching strategies. *Decision*, ???(??):???, Apr. 2019. 46
937. G. Leech, P. Rayson, and A. Wilson. *Word Frequencies in Written and Spoken English*. Pearson Education, 2001. 41, 178
938. L. Lefebvre and N. J. Boogert. Avian social learning. In M. D. Breed and J. Moore, editors, *Encyclopedia of Animal Behavior: volume 1*, pages 124–130. Oxford: Academic Press, July 2010. 68
939. J.-A. LeFevre and J. Liu. The role of experience in numerical skill: Multiplication performance in adults from Canada and China. *Mathematical Cognition*, 3(1):31–62, 1997. 46
940. G. E. Legge, T. A. Hooven, T. S. Klitz, J. S. Mansfield, and B. S. Tjan. Mr. Chips 2002: New insights from an ideal-observer model of reading. *Vision Research*, 42(18):2219–2234, Aug. 2002. 25
941. D. R. Lehman, R. O. Lempert, and R. E. Nisbett. The effects of graduate training on reasoning. *American Psychologist*, 43(6):431–442, 1988. 36
942. L. Lehmann, K. Aoki, and M. W. Feldman. On the number of independent cultural traits carried by individuals and populations. *Philosophical Transactions of The Royal Society B*, 366(1563):424–435, Feb. 2011. 68, 92
943. P. Lemaire and M. Fayol. When plausibility judgments supersede fact retrieval: The example of the odd-even effect on product verification. *Memory & Cognition*, 23(1):34–48, Feb. 1995. 44
944. P. Lennie. The cost of cortical computation. *Current Biology*, 13(6):493–497, Mar. 2003. 51
945. K. Lerman, X. Yan, and X.-Z. Wu. The "majority illusion" in social networks. *PLoS ONE*, 11(2):e0147617, Feb. 2016. 90
946. D. E. Levari, D. T. Gilbert, T. D. Wilson, B. Sievers, D. M. Amodio, and T. Wheatley. Prevalence-induced concept change in human judgment. *Science*, 360(6396):1465–1467, June 2018. 47
947. B. W. Leverett, R. G. G. Cattell, S. O. Hobbs, J. M. Newcomer, A. H. Reiner, B. R. Schatz, and W. A. Wulf. An overview of the production-quality compiler-compiler project. Technical Report CMU-CS-79-105, Carnegie Mellon University, Feb. 1979. 164
948. P. Lewicki, T. Hill, and E. Bizot. Acquisition of procedural knowledge about a pattern of stimuli that cannot be articulated. *Cognitive Psychology*, 20(1):24–37, Jan. 1988. 177
949. G. Lewis and P. Bajari. Incentives and adaptation: Evidence from highway procurement in Minnesota. Working Paper 17647, National Bureau of Economic Research, USA, Dec. 2011. 116
950. J. R. Lewis. Evaluation of procedures for adjusting problem-discovery rates estimated from small samples. *International Journal of Human-Computer Interaction*, 13(4):445–479, Dec. 2001. 158, 159
951. L. Li, Tegawendé, F. Bissyandé, Y. L. Traon, and J. Klein. Accessing inaccessible android APIs: An empirical study. In *International Conference on Software Maintenance and Evolution*, ICSME 2016, page ???, Oct. 2016. 106
952. X. Li. *Soft Error Modeling and Analysis for Microprocessors*. PhD thesis, University of Illinois at Urbana-Champaign, May 2008. 156
953. Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, 32(3):176–192, Mar. 2006. 118
954. Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a BlueGene/L prototype. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, DSN'05, pages 476–485, June 2005. 365
955. S. Lichtenstein and B. Fishhoff. Do those who know more also know more about how much they know? *Organizational Behavior and Human Performance*, 20:159–183, 1977. 50

956. Y. Lichtenstein and A. McDonnell. Pricing software development services. In *ECIS 2003 Proceedings*, 2003. 120
957. G. A. Liebchen and M. Shepperd. Data sets and data quality in software engineering. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, PROMISE'08, pages 39–44, May 2008. 357
958. J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, ICSE'10, pages 105–114, May 2010. 166, 167
959. J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. Scalable analysis of variable software. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE'13, pages 81–91, Aug. 2013. 154
960. F. Lieder, T. L. Griffiths, Q. J. M. Huys, and N. D. Goodman. The anchoring bias reflects rational use of cognitive resources. *Psychonomic Bulletin & Review*, 25(1):322–349, Feb. 2018. 20
961. J. H. Lienhard. *How Invention Begins: Echoes of Old Voices in the Rise of New Machines*. Oxford University Press, 2006. 86
962. B. P. Lientz, E. B. Swanson, and G. E. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, June 1978. 134
963. J. S. Light. When computers were women. *Technology and Culture*, 40(3):455–483, July 1999. 97
964. S. L. Lim. *Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, Aug. 2010. 122, 127
965. S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering*, 41(1):40–64, Jan. 2015. 95
966. B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th International Conference on Software Engineering*, ICSE'18, pages 94–104, May-June 2018. 330
967. D.-Y. Lin and I. Neamtiu. Collateral evolution of applications and databases. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution and Software Evolution (Evol) workshops*, IWPSE-Evol'09, pages 31–40, Aug. 2009. 188
968. L. C. H. Lin and N. Shen. GPL-3.0 in the Chinese intellectual property court in Beijing. *International Free and Open Source Software Law Review*, 10(1):1–7, 2018. 64
969. M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. API change and fault proneness: A threat to the success of Android apps. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 477–487, Aug. 2013. 144
970. K. R. Linberg. Software developer perceptions about software project failure: a case study. *The Journal of Systems and Software*, 49(2–3):177–192, Dec. 1999. 110
971. K. Lind and R. Heldal. A practical approach to size estimation of embedded software components. *IEEE Transactions on Software Engineering*, 38(5):993–1007, Sept.-Oct. 2012. 117, 118
972. R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150, Dec. 2004. 340
973. T. Little. Schedule estimation and uncertainty surrounding the cone of uncertainty. *IEEE Software*, 23(3):48–54, May 2006. 124, 125
974. D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental models and software maintenance. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, chapter 6, pages 80–98. Ablex Publishing Corporation, 1986. 168
975. S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Analysis of the Linux kernel evolution using code clone coverage. In *Fourth International Workshop on Mining Software Repositories*, MSR'07, pages 22–25, May 2007. 188
976. G. D. Logan. Shapes of reaction-time distributions and shapes of learning curves: A test of the instance theory of automaticity. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 18(5):883–914, 1992. 32
977. C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, and J. Vitek. DéjàVu: A map of code duplicates on GitHub. In *Conference on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA'17, page 84, Oct. 2017. 179, 180
978. I. Lorge and H. Solomon. Two models of group behavior in the solution of eureka-type problems. *Psychometrika*, 20(2):139–148, June 1955. 72
979. D. D. Loschelder, M. Friese, M. Schaefer, and A. D. Galinsky. The too-much-precision effect: When and why precise anchors backfire with experts. *Psychological Science*, 27(12):1573–1587, Oct. 2016. 116
980. P. Louridas, D. Spinellis, and V. Vlachos. Power laws in software. *ACM Transaction on Software Engineering and Methodology*, 18(1):1–26, Sept. 2008. 297
981. L. Lu, A. C. Arpacı-Dusseau, R. H. Arpacı-Dusseau, and S. Lu. A study of Linux file system evolution. In *11th USENIX Conference on File and Storage Technologies*, FAST'13, pages 31–44, Feb. 2013. 180, 197
982. J. D. Lucente. *On the Viability of Quantitative Assessment Methods in Software Engineering and Software Services*. PhD thesis, School of Engineering and Computer Science, University of Denver, Jan. 2015. 145
983. L. Lucia. *Ranking-Based Approaches for Localizing Faults*. PhD thesis, Singapore Management University, June 2014. 154
984. L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 26(2):172–219, Feb. 2014. 152
985. K. M. Lui and K. C. C. Chan. Pair programming productivity: Novice-novice vs. expert-expert. *International Journal of Human-Computer Studies*, 64(9):915–925, Sept. 2006. 33
986. P. Lukowicz, E. A. Heinz, L. Prechelt, and W. F. Tichy. Experimental evaluation in computer science: A quantitative study. Technical Report 17/94, University of Karlsruhe, Germany, Aug. 1994. 6
987. A. Lundqvist and D. Rodic. GNU/Linux distribution timeline. website, Oct. 2012. <http://futurist.se/gldt>. 88, 89
988. M. I. Lunesu. *Process Software Simulation Model of Lean-Kanban Approach*. PhD thesis, Department of Electrical and Electronic Engineering, University of Cagliari, Apr. 2013. 321, 322
989. Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 467–478, June 2014. 156
990. A. K. Luria. Towards the problem of the historical nature of psychological processes. *International Journal of Psychology*, 6(4):259–272, 1971. 17, 39
991. A. R. Luria. *The Mind of a Mnemonist*. Penguin Education, 1975. 31
992. B. Luthiger and C. Jungwirth. Pervasive fun. *First Monday*, 12(1), Jan. 2007. 284
993. R. Lutz. Analyzing software requirements errors in safety-critical, embedded systems. TR 92–27, Department of Computer Science, Iowa State University of Science and Technology, Aug. 1992. 152
994. W. J. Lynn III. A new approach for delivering information technology capabilities in the department of defense. Report to congress, Office of the Secretary of Defense, Nov. 2010. 123
995. W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu. How do developers fix cross-project correlated bugs? A case study on the GitHub scientific Python ecosystem. In *IEEE/ACM 39th International Conference on Software Engineering*, ICSE'17, pages 381–392, May 2017. 141
996. W. Ma, J.-C. S. Liu, and A. Forin. Design and testing of a cpu emulator. Technical Report MSR-TR-2009-155, Microsoft Research, Aug. 2009. 155
997. N. Macdonald. Computing services survey. *Computers and Automation*, 7(7):9–12, July 1958. 94
998. N. Macdonald. *Computer Census 1962-74*. Computers and People, 1974. 94
999. J. N. MacGregor. Short-term memory capacity: Limitation or optimization? *Psychological Review*, 94(1):107–108, 1987. 30

1000. A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for Android apps. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE'13, pages 224–234, Aug. 2013. [327](#), [328](#), [329](#)
1001. C. E. Mackenzie. *Coded Character Sets, History and Development*. Addison–Wesley, 1980. [96](#)
1002. D. MacKenzie. Computer-related accidental death: an empirical exploration. *Science and Public Policy*, 21(4):233–248, Aug. 1994. [142](#)
1003. I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *International Journal of Human-Computer Interaction*, 7(1):91–139, Mar. 1992. [53](#)
1004. R. J. Madachy. *Software Process Dynamics*. John Wiley & Sons, Inc, 2008. [115](#)
1005. A. Maddison. Business cycles, long waves and phases of capital development. In A. Maddison, editor, *Dynamic Forces in Capitalist Development: A Long-run Comparative View*, chapter 4, page ??? Oxford University Press, Oct. 1991. [5](#)
1006. W. T. Maddox and C. J. Bohil. Costs and benefits in perceptual categorization. *Memory & Cognition*, 28:597–615, 2000. [36](#)
1007. M. Magidin and E. Viso. On the experiments in algorithm dynamics. Technical Report UAMR0853, Universidad Autónoma Metropolitana-Iztapalapa, México, Oct. 1976. [178](#)
1008. T. Maillart, M. Zhao, J. Grossklags, and J. Chuang. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity*, 3(2):81–90, June 2017. [98](#), [144](#)
1009. S. Majd and R. S. Pindyck. Time to build, option value, and investment decisions. *Journal of Financial Economics*, 18(1):7–27, Mar. 1987. [60](#)
1010. V. Makarov. 2016 Export of Russian software development industry. Annual Survey 13-th, RUSSOFT Association, Aug. 2016. [102](#)
1011. V. N. Makarov. SPEC benchmark page. <http://vmakarov.fedorapeople.org/spec/index.html>, July 2014. [353](#), [354](#)
1012. B. A. Malloy and J. F. Power. Quantifying the transition from Python 2 to 3: An empirical study of Python applications. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM'17, pages 314–323, Dec. 2017. [181](#)
1013. M. Mangel and F. J. Samaniego. Abraham Wald's work on aircraft survivability. *Journal of the American Statistical Association*, 79(386):259–267, June 1984. [233](#)
1014. Manpower. Computers in offices. Studies No. 4, Manpower Research Unit, Ministry of Labour, G.B., 1965. [81](#)
1015. M. V. Mäntylä and J. Itkonen. How are software defects found? The role of implicit defect detection, individual responsibility, documents, and knowledge. *Information and Software Technology*, 56(12):1597–1612, Dec. 2014. [161](#)
1016. A. Marchand. The power of an installed base to combat lifecycle decline: The case of video games. *International Journal of Research in Marketing*, 33(1):140–154, Mar. 2016. [77](#)
1017. J. N. Marewski and L. J. Schooler. Cognitive niches: An ecological model of strategy selection. *Psychological Review*, 118(3):292–437, 2011. [48](#)
1018. B. H. Margolin, R. P. Parmelee, and M. Schatzoff. Analysis of free-storage algorithms. *IBM Systems Journal*, 10(4):283–304, 1971. [182](#)
1019. P. Marinescu, P. Hosek, and C. Cadar. COVRIG: A framework for the analysis of code, test, and coverage evolution in real software. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA'14, pages 93–104, July 2014. [160](#)
1020. F. Marotta-Wurgler. What's in a standard form contract? An empirical analysis of software license agreements. *Journal of Empirical Legal Studies*, 7(4):677–713, Dec. 2007. [120](#)
1021. F. Marotta-Wurgler. Will increased disclosure help? Evaluating the recommendations of the ALI's "principles of the law of software contracts". *University of Chicago Law Review*, 78(1), 2011. [63](#)
1022. D. H. Martin, J. R. Cordy, B. Adams, and G. Antoniol. Make it simple-An empirical analysis of GNU make feature use in open source projects. In *IEEE 23rd International Conference on Program Comprehension*, pages 207–217, May 2015. [187](#)
1023. F. Martineau. PNFG: A framework for computer game narrative analysis. Thesis (m.s.), School of Computer Science, McGill University, June 2006. [182](#)
1024. A. G. Martínez. *Chaos Monkeys: Inside The Silicon Valley Money Machine*. Ebury Press, 2016. [83](#)
1025. E. Masanet, A. Shehabi, J. Liang, L. Ramakrishnan, X. Ma, V. Hendrix, B. Walker, and P. Mantha. The energy efficiency potential of cloud-based software: A U.S. case study. LBNL Paper LBNL-6298E, Lawrence Berkeley National Laboratory, June 2013. [84](#)
1026. F. Massacci, S. Neuhaus, and V. H. Nguyen. After-life vulnerabilities: A study on Firefox evolution, its vulnerabilities, and fixes. In *Proceedings of the Third international conference on Engineering secure software and systems*, ESSoS'11, pages 195–208, Feb. 2011. [149](#), [150](#), [151](#), [152](#)
1027. F. Mathy, M. Chekaf, and N. Cowan. Simple and complex working memory tasks allow similar benefits of information compression. *Journal of Cognition*, 1(31):1–12, May 2018. [29](#)
1028. E. Matias, I. S. MacKenzie, and W. Buxton. One-handed touch-typing on a QWERTY keyboard. *International Journal of Human-Computer Interaction*, 11:1–27, 1996. [19](#)
1029. C. Mayer, S. Hamenberg, R. Robbes, É. Tanter, and A. Stefik. An empirical study of the influence of static type systems on the usability of undocumented software. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, OOPSLA'12, pages 683–702, Oct. 2012. [168](#)
1030. D. Mazinanian, A. Ketkar, N. Tsantalis, and D. Dig. Understanding the use of lambda expressions in Java. In *Proceedings of the ACM on Programming Languages*, OOPSLA'17, page 85, Oct. 2017. [181](#)
1031. A. Mazouz. *An Empirical Study of Program Performance of OpenMP Applications on Multicore Platforms*. PhD thesis, Université de Versailles-Saint Quentin en Yvelines, Dec. 2013. [352](#)
1032. M. Mazzucato. Risk, variety and volatility in the PC industry: New economy or early life-cycle? In *NY Federal Reserve Bank conference on "Productivity Growth: A New Era?"*, Nov. 2001. [91](#)
1033. D. F. McAllister and M. A. Vouk. Experiments in fault tolerant software reliability. Technical Report 5 – NAG-1-667, North Carolina State University, Apr. 1989. [118](#), [161](#), [162](#)
1034. T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, Dec. 1976. [178](#)
1035. J. C. McCallum. Historical cost of computer memory and storage. <http://www.jcmi.com>, July 2016. [1](#)
1036. S. McCloud. *Understanding Comics*. HarperPerennial, 1993. [206](#)
1037. S. McConnell. *Code Complete*. Microsoft Press, 1993. [166](#)
1038. M. H. McCormack. *The Terrible Truth about Lawyers*. Beech Tree books, William Morrow, 1987. [120](#)
1039. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4):125–180, June 2001. [340](#)
1040. R. R. McCrae and P. T. Costa, Jr. Reinterpreting the Myers-Briggs type indicator from the perspective of the five-factor model of personality. *Journal of Personality*, 57(1):17–40, Mar. 1989. [47](#)
1041. B. D. McCullough. Microsoft Excel's 'Not The Wichmann-Hill' random number generator. *Computational Statistics & Data Analysis*, 52:4587–4593, 2008. [155](#)
1042. B. D. McCullough and D. A. Heiser. On the accuracy of statistical procedures in Microsoft Excel 2007. *Computational Statistics & Data Analysis*, 52:4570–4578, 2008. [13](#)
1043. R. McElreath and R. Boyd. *Mathematical Models of Social Evolution: A Guide for the Perplexed*. The University of Chicago Press, 2008. [68](#), [90](#)
1044. D. McFadden. Rationality for economists? *Journal of Risk and Uncertainty*, 19:73–105, 1999. [48](#)
1045. R. W. McGee. *Accounting for Software in the United States*. PhD thesis, School of Industrial and Business Studies, University of Warwick, Apr. 1986. [74](#)
1046. B. McGonigle, M. Chalmers, and A. Dickinson. Concurrent disjoint and reciprocal classification by Cebus apella in seriation tasks: evidence for hierarchical organization. *Animal Cognition*, 6(3):185–197, Sept. 2003. [36](#)

1047. S. McJohn. The GPL meets the UCC: Does free software come with a warranty of no infringement? *Journal of High Technology Law*, XV(1):1–62, 2014. [64](#)
1048. K. B. McKeithen, J. S. Reitman, H. H. Ruster, and S. C. Hirtle. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13(3):307–325, July 1981. [35](#), [36](#)
1049. J. McManus and T. Wood-Harper. Understanding the sources of information systems project failure: A study in IS project failure. *Management Services*, 51(3):38–43, Aug. 2007. [112](#)
1050. D. S. McNamara and J. Magliano. Toward a comprehensive model of comprehension. In B. Ross, editor, *The Psychology of Learning and Motivation*, Vol. 51, chapter 9, pages 297–384. Academic Press, Nov. 2009. [172](#)
1051. T. P. McNamara, J. K. Hardy, and S. C. Hirtle. Subjective hierarchies in spatial memory. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 15(2):211–227, 1989. [26](#)
1052. J. McNerney, J. D. Farmer, S. Redner, and J. E. Trancik. Role of design complexity in technology improvement. *Proceedings of the National Academy of Sciences*, 108(22):9008–9013, May 2011. [69](#)
1053. D. L. McNicol. Influences on the timing and frequency of cancellations and truncations of major defense acquisition programs. IDA Paper P-8280, Institute for Defense Analyses, Mar. 2017. [112](#)
1054. I. McPhee. Customs' cargo management re-engineering project: Australian customs service. Audit Report No.24 2006-07, Australian National Audit Office, Aug. 2007. [110](#)
1055. J. H. McWhorter. The world's simplest grammars are creole grammars. *Linguistic Typology*, 5(2-3):125–166, 2001. [181](#)
1056. A. D. Meacham. *Data Processing Equipment Encyclopedia: Electronic Devices*, volume 2. Gille Associates, Inc., 1962. [102](#), [345](#)
1057. F. Mechner. Probability relations within response sequences under ratio reinforcement. *Journal of the Experimental Analysis of Behavior*, 1(2):109–121, Apr. 1958. [16](#)
1058. F. Medeiros, C. Kästner, M. Ribeiro, S. Nadi, and R. Gheyi. The love/hate relationship with the C preprocessor: An interview study. In *Proceedings of the 29th European Conference on Object-Oriented Programming*, ECOOP’15, pages 495–518, July 2015. [166](#)
1059. M. Meeks. German comments in LibreOffice. website, Apr. 2017. <https://people.gnome.org/~michael/data/2015-08-01-5.5-data.ods>. [96](#)
1060. M. Meeter, J. M. J. Murre, and S. M. J. Janssen. Remembering the news: Modeling retention data from a study with 14,000 participants. *Memory & Cognition*, 33(5):793–810, July 2004. [31](#), [32](#)
1061. M. Mehrara and T. Austin. Exploiting selective placement for low-cost memory protection. *ACM Transactions on Architecture and Code Optimization*, 5(3):1–24, Nov. 2008. [156](#)
1062. L. K. Melhus and R. E. Jensen. Measurement bias from address aliasing. In *Eleventh International Workshop on Automatic Performance Tuning*, iWAPT 2016, pages 1506–1515, May 2016. [349](#)
1063. R. Meloia, G. Pinto, L. Baiser, M. Mattos, I. Polato, I. S. Wiese, and D. M. German. Understanding the usage, impact, and adoption of non-OSI approved licenses. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR’18, pages 270–280, May 2018. [63](#), [64](#)
1064. D. Mendez, B. Baudry, and M. Monperrus. Empirical evidence of large-scale diversity in API usage of object-oriented software. In *International Conference on Source Code Analysis and Manipulation*, SCAM’13, pages 43–52, Apr. 2013. [185](#), [186](#)
1065. H. Mengistu, J. Huizinga, J.-B. Mouret, and J. Clune. The evolutionary origins of hierarchy. *PLOS Computational Biology*, 12(6):e1004829, June 2016. [185](#)
1066. H. Mercier and D. Sperber. Why do humans reason? Arguments for an argumentative theory. *Behavioral and Brain Sciences*, 34(2):57–111, Apr. 2011. [39](#)
1067. H. Mercier and D. Sperber. Why do humans reason? Arguments for an argumentative theory. HAL Id: hal-00904097, HAL archives-ouvertes.fr, Nov. 2013. [39](#)
1068. R. C. Merkle. Energy limits to the computational power of the human brain. *Foresight Update*, 6, Aug. 1989. [51](#)
1069. E. W. Merrow, L. McDonnel, and R. W. Argüden. Understanding the outcomes of megaprojects: A quantitative analysis of very large civilian projects. Report R-3560-PSSP, The RAND Corporation, Mar. 1988. [113](#)
1070. S. Mertens and C. Baethge. The virtues of correct citation. *Deutsches Ärzteblatt International*, 108(33):550–552, Apr. 2011. [138](#)
1071. A. Mesoudi. Cultural evolution: A review of theory, findings and controversies. *Evolutionary Biology*, 43(4):481–497, Dec. 2016. [67](#)
1072. L. A. Meyerovich and A. Rabkin. How not to survey developers and repositories: Experiences analyzing language adoption. In *Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU’12, pages 7–16, Oct. 2012. [104](#)
1073. T. Micceri. The unicorn, the normal curve, and other improbable creatures. *Psychological Bulletin*, 105(1):156–166, Apr. 1989. [231](#)
1074. S. E. Michalak, A. J. DuBois, C. B. Storlie, H. M. Quinn, W. N. Rust, D. H. DuBois, D. G. Modl, A. Manuzzato, and S. P. Blanchard. Assessment of the impact of cosmic-ray-induced neutrons on hardware in the Roadrunner supercomputer. *IEEE Transactions on Device and Materials Reliability*, 12(2):445–454, May 2012. [156](#)
1075. J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, The Google Books Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 34(6104):176–182, Jan. 2011. [103](#), [365](#)
1076. Microsoft server protocol documentation. website, 2015. <http://www.microsoft.com>. [73](#), [107](#), [200](#), [201](#)
1077. S. Milgram. *Obedience to Authority*. McGraw-Hill, 1974. [49](#)
1078. S. Milgram, L. Bickman, and L. Berkowitz. Note on the drawing power of crowds of different size. *Journal of Personality and Social Psychology*, 13(2):79–82, 1969. [67](#)
1079. A. Mili, S. F. Chmiel, R. Gottumukkala, and L. Zhang. Managing software reuse economics: An integrated ROI-based model. *Annals of Software Engineering*, 11(1):175–218, Nov. 2001. [117](#)
1080. K. Milis. *Success factors for ICT projects: Empirical research, utilising qualitative and quantitative approaches (incl. Bayesian networks, Probabilistic feature models)*. PhD thesis, Toegepaste Economische Wetenschappen, Limburgs Universitat Centrum, Dec. 2002. [110](#)
1081. D. M. Miller. Application of halstead’s timing model to predict the compilation time of Ada compilers. Thesis (m.s.), School of Engineering of the Air Force Institute of Technology, Air University, Dec. 1986. [165](#)
1082. D. R. Miller. Exponential order statistic models of software reliability growth. NASA Contractor Report 3909, NASA Langley Research Center, July 1985. [141](#), [148](#)
1083. G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956. [27](#), [341](#)
1084. G. A. Miller and S. Isard. Free recall of self-embedded English sentences. *Information and Control*, 7:292–303, 1964. [28](#)
1085. L. A. Miller. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, 29(2):184–215, 1981. [168](#)
1086. S. J. Miller and M. J. Nigrini. Order statistics and Benford’s law. *International Journal of Mathematics and Mathematical Sciences*, 2008, 2008. [366](#)
1087. W. R. Miller and M. Sanchez-Craig. How to have a high success rate in treatment: advice for evaluators of alcoholism programs. *Addiction*, 91(6):779–785, Apr. 1996. [339](#)
1088. S. MINAKAWA, T. HIRATA, K. MASAME, H. OKADA, and K. MARUYAMA. A psychological analysis on the meaning of "reliance". *Tohoku Psychologica Folia*, 46(1-4):111–117, Apr. 1987. [142](#)
1089. C. H. Mireles. *Marketing Modeling for New Products*. PhD thesis, Erasmus University, Rotterdam, June 2010. [78](#)
1090. MISRA. *MISRA-C:2004 Guidelines for the Use of the C Language in Vehicle Based Software*. Motor Industry Research Association, 2004. [142](#), [166](#)
1091. MISRA. *MISRA-C++:2008 Guidelines for the Use of the C++ Language in Critical Systems*. Motor Industry Research Association, June 2008. [142](#)
1092. K. E. Mitchell. The copyleft bust up: loopholes, licenses, and realpolitik in open source. /dev/lawyer blog, Nov. 2018. <https://writing.kemitchell.com/2018/11/04/Copyleft-Bust-Up.html>. [62](#)
1093. K. Mitropoulou. *Performance Optimizations for Compiler-based Error Detection*. PhD thesis, School of Informatics, University of Edinburgh, Oct. 2014. [156](#)

1094. S. Mittal. A survey of architectural techniques for managing process variation. *ACM Computing Surveys*, 48(4), May 2016. [347](#)
1095. M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2003. [217](#)
1096. M. Mitzenmacher. Dynamic models for file sizes and double Pareto distributions. *Internet Mathematics*, 1(3):305–333, Apr. 2004. [205](#), [225](#)
1097. E. Miu, N. Gulley, K. N. Laland, and L. Rendel. Innovation and cumulative culture through tweaks and leaps in online programming contests. *Nature Communications*, 9(2321), June 2018. [96](#)
1098. O. Mlouki. On the detection of licenses violations in the Android ecosystem. Thesis (m.s.), Université de Montréal, Apr. 2016. [63](#)
1099. A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance, ICSM’00*, pages 120–130, Oct. 2000. [134](#)
1100. A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2), Apr.–June 2000. [33](#)
1101. S. N. Mohanty. Software cost estimation: Present and future. *Software—Practice and Experience*, 11(2):103–121, Feb. 1981. [115](#)
1102. T. Moher and G. M. Schneider. Methods for improving controlled experimentation in software engineering. In *Proceedings of the 5<sup>th</sup> International Conference on Software Engineering*, pages 224–233. IEEE Computer Society, Mar. 1981. [340](#)
1103. K. Moløkken-Østvold and K. M. Furulund. The relationship between customer collaboration and software project overruns. In *2007 Agile Conference, AGILE*, pages 72–83, Aug. 2007. [289](#)
1104. K. Moløkken-Østvold, M. Jørgensen, S. S. Tanilkan, H. Gallis, A. C. Lien, and S. E. Hove. A survey on software estimation in the norwegian industry. In *Proceedings 10th International Symposium on Software Metrics*, pages 208–219, Sept. 2004. [117](#)
1105. G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, Apr. 1965. [4](#)
1106. A. C. Morgan, D. J. Economou, S. F. Way, and A. Clauset. Prestige drives epistemic inequality in the diffusion of scientific ideas. In *eprint arXiv:cs.SI/1805.09966*, Oct. 2018. [67](#)
1107. T. Moscibroda and R. Oshman. Resilience of mutual exclusion algorithms to transient memory faults. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing, PODC’11*, pages 69–78, June 2011. [156](#)
1108. C. Motta. Analysing the evolution of system requirements – A case study of AUTOSAR at Volvo car group. Thesis (m.s.), Department of Computer Science and Engineering, Gothenburg, June 2016. [95](#)
1109. J. F. Motz. In re microsoft corporation antitrust litigation \* sun microsystems, inc. v. microsoft corporation \* mdl 1332 \* civil no. jfm-02-2739. Opinion, UNITED STATES DISTRICT COURT FOR THE DISTRICT OF MARYLAND, 2002. [100](#)
1110. Y. Moy and A. Wallenburg. Tokeneer: Beyond formal program verification. In *Proceedings of the 5th International Congress on Embedded Real Time Software and Systems, ERTS<sup>2</sup> 2010*, May 2010. [139](#)
1111. S. T. Mueller and A. Krawitz. Reconsidering the two-second decay hypothesis in verbal working memory. *Journal of Mathematical Psychology*, 53(1):14–25, Feb. 2009. [27](#)
1112. S. T. Mueller and C. T. Weidemann. Alphabetic letter identification: Effects of perceptability, similarity, and bias. *Acta Psychologica*, 139(1):19–37, Jan. 2012. [19](#), [176](#)
1113. D. Mulcahy, B. Weeks, and H. S. Bradley. "WE HAVE MET THE ENEMY... AND HE IS US" Lessons from twenty years of the Kauffman Foundation's investments in venture capital funds and the triumph of hope over experience. Report, Ewing Marion Kauffman Foundation, May 2012. [83](#)
1114. C. W. Mulford and S. Misra. Capitalization of software development costs: Accounting practices in the software industry, 2014 and 2015. Technical report, Georgia Tech College of Management, Jan. 2016. [56](#), [74](#), [75](#)
1115. C. W. Mulford and J. Roberts. Capitalization of software development costs: A survey of accounting practices in the software industry. Technical report, Georgia Tech College of Management, May 2006. [75](#)
1116. M. M. Müller and A. Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12(6):593–615, 2007. [194](#), [340](#), [359](#)
1117. J. Mulligan and B. Patrovsky. *Developing Online Games: An Insider’s Guide*. New Riders Publishing, 2003. [121](#)
1118. D. Muna, M. Alexander, A. Allen, R. Ashley, D. Asmus, R. Azollini, M. Bannister, R. Beaton, A. Benson, G. B. Berriman, M. Bilicki, P. Boyce, J. Bridge, J. Cami, E. Cangi, X. Chen, N. Christiny, C. Clark, M. Collins, J. Comparat, N. Cook, D. Croton, I. D. Davids, É. Depagne, J. Donor, L. A. dos Santos, S. Douglas, A. Du, M. Durbin, D. Erb, D. Faes, J. G. Fernández-Trincado, A. Foley, S. Fotopoulou, S. Frimann, P. Frinchaboy, R. Garcia-Dias, A. Gawryszczak, E. George, S. Gonzalez, K. Gordon, N. Gorgone, C. Gosmeyer, K. Grasha, P. Greenfield, R. Grellmann, J. Guillotin, M. Gurwell, M. Haas, A. Hagen, D. Haggard, T. Haines, P. Hall, W. Hellwing, E. C. Herenz, S. Hinton, R. Hlozek, J. Hoffman, D. Holman, B. W. Holwerda, A. Horton, C. Hummels, D. Jacobs, J. J. Jensen, D. Jones, A. Karick, L. Kelley, M. Kenworthy, B. Kitchener, D. Klaes, S. Kohn, P. Konorski, C. Krawczyk, K. Kuehn, T. Kuutma, M. T. Lam, R. Lane, J. Liske, D. Lopez-Camara, K. Mack, S. Mangham, Q. Mao, D. J. E. Marsh, C. Mateu, L. Maurin, J. McCormac, I. Momcheva, H. Monteiro, M. Mueller, R. Munoz, R. Naidu, N. Nelson, C. Nitschelm, C. North, J. Nunez-Iglesias, S. Ogaz, R. Owen, J. Parejko, V. Patrício, J. Pepper, M. Perrin, T. Pickering, J. Piscionere, R. Pogge, R. Poleski, A. Pourtsidou, A. M. Price-Whelan, M. L. Rawls, S. Read, G. Rees, H. Rein, T. Rice, S. Riemer-Sørensen, N. Rusomarov, S. F. Sanchez, M. Santander-García, G. Sarid, W. Schoenell, A. Scholz, R. L. Schuhmann, W. Schuster, P. Scicluna, M. Seidel, L. Shao, P. Sharma, A. Shulevski, D. Shupe, C. Sifón, B. Simmons, M. Sinha, I. Skillen, B. Soergel, T. Spriggs, S. Srinivasan, A. Stevens, O. Streicher, E. Suchtya, J. Tan, O. G. Telford, R. Thomas, C. Tonini, G. Tremblay, S. Tuttle, T. Urrutia, S. Vaughan, M. Verdugo, A. Wagner, J. Walawender, A. Wetzel, K. Willett, P. K. G. Williams, G. Yang, G. Zhu, and A. Zonca. The Astropy problem. In *eprint arXiv:astro-ph.IM/1610.03159*, Oct. 2016. [67](#), [112](#)
1119. B. B. Murdoch, Jr. The serial position effect of free recall. *Journal of Experimental Psychology*, 64(5):482–488, 1962. [30](#)
1120. J. M. J. Murre and A. G. Chessa. Power laws from individual differences in learning and forgetting: mathematical analyses. *Psychonomic Bulletin & Review*, 18(3):592–597, June 2011. [225](#)
1121. J. M. J. Murre and J. Dros. Replication and analysis of Ebbinghaus' forgetting curve. *PLoS ONE*, 10(7):e0120644, July 2015. [31](#)
1122. P. Murrell. *R Graphics*. Chapman & Hall/CRC, 1st edition, 2006. [203](#)
1123. M. Muthukrishna, B. W. Shulman, V. Vasilescu, and J. Henrich. Sociability influences cultural complexity. *Proceedings of the Royal Society B: Biological Sciences*, 281(1774), Nov. 2013. [69](#)
1124. L. H. Mutuel. Single event effects mitigation techniques report. Final Report DOT/FAA/TC-15/62, U.S. Department of Transportation, Federal Aviation Administration, Feb. 2016. [155](#)
1125. G. J. Myers. A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM*, 21(9):760–768, Sept. 1978. [158](#)
1126. T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. We have it easy, but do we have it right? In *International Symposium on Parallel and Distributed Processing, IPDPS 2008*, pages 1–7, Apr. 2008. [349](#), [351](#)
1127. M. Nagappan, R. Robbes, Y. Kamei, É. Tanter, S. McIntosh, A. Mockus, and A. E. Hassan. An empirical study of goto in C code from GitHub repositories. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE’15*, pages 404–414, Aug.–Sept. 2015. [183](#)
1128. M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013*, pages 466–476, Aug. 2013. [233](#)
1129. N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *IEEE 21st International Symposium on Software Reliability Engineering, ISSRE’10*, pages 309–318, Nov. 2010. [293](#)
1130. P. M. Nagel, F. W. Scholz, and J. A. Skrivan. Software reliability: Additional investigations into modeling with replicated experiments. NASA Contractor Report 172378, Boeing Computer Services Company, Space and Military Applications Division, June 1984. [146](#)

1131. P. M. Nagel and J. A. Skrivan. Software reliability: Repetitive run experimentation and modeling. NASA Contractor Report 165836, Boeing Computer Services Company, Space and Military Applications Division, Feb. 1982. 146, 147
1132. T. Nagle, J. Hogan, and J. Zale. *The Strategy and Tactics of Pricing*. Pearson, fifth edition, 2015. 76
1133. J. S. Nairne. The loss of positional certainty in long-term memory. *Psychological Science*, 3(2):199–202, May 1992. 30
1134. J. Nandhakumar and D. E. Avison. The fiction of methodological development: a field study of information systems development. *Information Technology & People*, 12(2):176–191, Feb. 1999. 123
1135. S. Nanz and C. A. Furia. A comparative study of programming languages in Rosetta Code. In *eprint arXiv:cs.SE/1409.0252v1*, Aug. 2014. 167
1136. M. B. Nathanson. Desperately seeking mathematical truth. *Notices of the AMS*, 55(7):773–773, Aug. 2008. 138
1137. P. Naur and B. Randell. Software engineering report on a conference sponsored by the NATO science committee. Technical report, NATO, Jan. 1969. 5, 102
1138. A. D. Navarro and E. Fantino. The sunk cost effect in pigeons and humans. *Journal of the Experimental Analysis of Behavior*, 83(1):1–13, Jan. 2005. 53
1139. D. J. Navarro and A. F. Perfors. Hypothesis generation, sparse categories, and the positive test strategy. *Psychological Review*, 118(1):120–134, Jan. 2011. 21
1140. I. Neamtiu, J. S. Foster, and M. Hicks. Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, pages 1–5, May 2005. 185
1141. S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig. Is it dangerous to use version control histories to study source code evolution? In *Proceedings of the 26th European conference on Object-Oriented Programming*, ECOOP’12, pages 79–103, June 2012. 361
1142. D. L. Nelson, C. L. McEvoy, and T. A. Schreiber. The university of South Florida word association, rhyme and word fragment norms. [w3.usf.edu/FreeAssociation](http://w3.usf.edu/FreeAssociation), 1998. 175
1143. E. A. Nelson. Management handbook for the estimation of computer programming costs. Technical Documentary Report ESD-TDR-67-66, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Oct. 1966. 114
1144. D. A. Nembhard and N. Osothsilp. An empirical comparison of forgetting models. *IEEE Transactions on Engineering Management*, 48(3):283–291, Aug. 2001. 69
1145. R. E. NeSmith II. A study of software maintenance costs of Air Force large scale computer systems. Thesis (m.s.), School of Systems and Logistics, Air Force Institute of Technology, Air University, Sept. 1986. 111, 282
1146. D. Nettle. Explaining global patterns of language diversity. *Journal of Anthropological Archaeology*, 17(4):354–374, Dec. 1998. 102
1147. B. New, L. Ferrand, C. Pallier, and M. Brysbaert. Reexamining the word length effect in visual word recognition: New evidence from the English lexicon project. *Psychonomic Bulletin & Review*, 13(1):45–52, Feb. 2006. 175
1148. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1991. 16
1149. A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the power law of practice. Technical report, Carnegie Mellon University, Aug. 1982. 32
1150. S. E. Newstead and K. R. Coventry. The role of context and functionality in the interpretation of quantifiers. *European Journal of Cognitive Psychology*, 12(2):243–259, June 2000. 45
1151. G. Nezlek and G. DeHondt. An empirical investigation of gender wage differences in information systems occupations: 1991–2008. In *Proceedings of the 43rd Hawaii International Conference on System Sciences—2010*, pages 4059–4068, Jan. 2010. 65
1152. T. H. D. Nguyen, B. Adams, and A. E. Hassan. A case study of bias in bug-fix datasets. In *17th Working Conference on Reverse Engineering*, WCRE’10, pages 259–268, Oct. 2010. 141
1153. V. H. Nguyen and F. Massacci. The (un)reliability of NVD vulnerable versions data: an empirical experiment on Google Chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS’13, pages 493–498, May 2013. 141
1154. T. Nicholas. *VC: An American History*. Harvard University Press, June 2019. 83
1155. T. Nichols. A penny saved: Psychological pricing. website, Oct. 2013. <http://blog.gumroad.com/post/64417917582/a-penny-saved-psychological-pricing>. 77
1156. J. Nielsen and T. K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERCHI ’93 conference on Human factors in computing systems*, INTERCHI’93, pages 206–213, Apr. 1993. 158
1157. E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs. In *Proceedings of the sixth conference on Computer systems*, EuroSys’11, pages 343–356, Apr. 2011. 256
1158. M. J. Nigrini and S. J. Miller. Data diagnostics using second order tests of Benford’s law. *Auditing: A Journal of Practice and Theory*, 28(2):305–324, June 2009. 366
1159. D. E. Nikonov and I. A. Young. Overview of beyond-CMOS devices and a uniform methodology for their benchmarking. In *eprint arXiv:cond-mat.mes-hall/1302.0244*, Feb. 2013. 348
1160. J. Ninio and K. A. Stevens. Variations on the Hermann grid: an extinction illusion. *Perception*, 29(10):1209–1217, Oct. 2000. 173
1161. R. E. Nisbett, D. H. Krantz, C. Jepson, and Z. Kunda. The use of statistical heuristics in everyday inductive reasoning. *Psychological Review*, 90(4):339–363, 1983. 36
1162. NIST???. National vulnerability database. <https://nvd.nist.gov>, Dec. 2014. 141, 360
1163. G. P.-C. no and D. M. German. Software patents: A replication study. In *Proceedings of the 11th International Symposium on Open Collaboration*, OpenSym’15, pages 5:1–5:4, Aug. 2015. 62
1164. S. Nørby. Why forget? On the adaptive value of memory loss. *Perspectives on Psychological Science*, 10(5):551–578, Sept. 2015. 31
1165. P. V. Norden. Resource usage and network planning techniques. In B. V. Dean, editor, *Operations Research in Research and Development*, chapter 5, pages 149–169. John Wiley & Sons, Inc, 1963. 115
1166. W. D. Nordhaus. The progress of computing. Cowles Foundation Discussion Paper No. 1324, Yale University, Sept. 2001. 1
1167. J. A. Norton and F. M. Bass. A diffusion theory model of adoption and substitution for successive generations of high-technology products. *Management Science*, 33(9):1069–1086, Sept. 1987. 78
1168. M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. The Belknap press of Harvard University press, 2006. 90
1169. M. A. Nowak and K. Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393(6685):573–577, June 1998. 71
1170. D. Nowroth, I. Polian, and B. Becker. A study of cognitive resilience in a JPEG compressor. In *IEEE International Dependable Systems and Networks With FTCS and DCC*, DSN 2008, pages 32–41, June 2008. 155
1171. H.-C. Nuerk, G. Wood, and K. Willmes. The universal snarc effect: The association between number magnitude and space is amodal. *Experimental Psychology*, 52(3):187–194, 2005. 19
1172. R. E. Núñez. No innate number line in the human brain. *Journal of Cross-Cultural Psychology*, 42(4):651–668, 2011. 43
1173. J. M. Nuttin, Jr. Affective consequence of mere ownership: The name letter effect in twelve European languages. *European Journal of Social Psychology*, 17:381–402, 1987. 95
1174. NVIDIA. *CUDA CUBLAS Library*. NVIDIA Corporation, CA, USA, 3.1 edition, Aug. 2010. 341
1175. M. Oaksford and N. Chater. A rational analysis of the selection task as optimal data selection. *Psychological Review*, 101(4):608–631, 1994. 40
1176. K. Oberauer, S. Lewandowsky, E. Awh, G. D. A. Brown, A. Conway, N. Cowan, C. Donkin, S. Farrell, G. J. Hitch, M. Hurlstone, W. J. Ma, C. C. Morey, D. E. Nee, J. Schweppé, E. Vergauwe, and G. Ward. Benchmarks for models of short term and working memory. *Psychological Bulletin*, 144(9):885–958, Sept. 2018. 27
1177. K. Oberauer, H.-M. Süß, O. Wilhelm, and W. W. Wittmann. The multiple faces of working memory: Storage, processing, supervision, and coordination. *Intelligence*, 31(2):167–193, Mar.-Apr. 2003. 27

1178. O. O. Odeh, A. M. Featherstone, and J. S. Bergtold. Reliability of statistical software. *American Journal of Agricultural Economics*, 92(5):1472–1489, Sept. 2010. [12](#)
1179. OECD. *OECD Digital Economy Outlook 2015*. OECD Publishing, 2015. [56](#)
1180. Office for National Statistics, UK. GFCF estimates for computer software purchases, own account computer software. website, June 2017. <http://www.ons.gov.uk/ons/rel/bus-invest/business-investment/index.html>. [4](#), [94](#)
1181. C. Ogden. Killed by Google. <https://killedbygoogle.com>, Nov. 2018. [59](#), [89](#), [90](#)
1182. S. Ogilvie. *The European Guilds: An Economic Analysis*. Princeton University Press, Feb. 2019. [71](#)
1183. S. Ohlsson. The learning curve for writing books: Evidence from Professor Asimov. *Psychological Science*, 3(6):380–382, Nov. 1992. [35](#)
1184. H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502–505, May 2006. [71](#)
1185. P. Oladimeji. Devices, errors and improving interaction design-A case study using an infusion pump. Thesis (m.res.), Department of Computer Science, Swansea University, Oct. 2008. [226](#)
1186. A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN’07, pages 575–584, June 2007. [157](#)
1187. P. Oliver. Experiences in building and using compiler validation systems. In R. Merwin and J. Zanca, editors, *AFIPS Conference Proceedings*, pages 1051–1057. AFIPS Press, June 1979. [160](#)
1188. O\*NET OnLine. website, July 2019. <https://www.onetonline.org>. [97](#)
1189. T. Open Group. The Austin common standards revision group. <http://austingroupbugs.net>, July 2017. [153](#)
1190. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, Aug. 2015. [9](#)
1191. OpenCorporates. UK registered company data. <https://opencorporates.com/>, Mar. 2015. [97](#)
1192. OpenRefine. website, Oct. 2014. <http://openrefine.org>. [358](#)
1193. OpenSignal. Android fragmentation visualized (august 2015). Technical Report ???, OpenSignal, Aug. 2015. [203](#), [204](#), [359](#)
1194. A. Orlitsky, A. T. Suresh, and Y. Wu. Optimal prediction of the number of unseen species. *PNAS*, 113(47):13283–13288, Nov. 2016. [93](#)
1195. N. Osaka. Eye fixation and saccade during kana and kanji text reading: Comparison of English and Japanese text processing. *Bulletin of the Psychonomic Society*, 27(6):548–550, 1989. [24](#)
1196. H. Osman, M. Leuenberger, M. Lungu, and O. Nierstrasz. Tracking null checks in open-source Java systems. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER’16, pages 304–313, Mar. 2016. [184](#)
1197. E. Ostrom, J. Walker, and R. Gardner. Covenants with and without a sword: Self-Governance is possible. *The American Political Science Review*, 86(2):404–417, June 1992. [71](#)
1198. L. M. Ottenstein, V. B. Schneider, and M. H. Halstead. Predicting the number of bugs expected in a program module. Technical Report CSD-TR 205, Purdue University, Oct. 1976. [178](#)
1199. M. A. Oumaziz, A. Charpentier, J.-R. Falleri, and X. Blanc. Documentation reuse: Hot or not? An empirical study. In *16th International Conference on Software Reuse*, ICSR 2017, pages 12–27, May 2017. [152](#)
1200. G. L. Ourada. Software cost estimating models: A calibration, validation, and comparison. Thesis (m.s.), Air Force Institute of Technology, Air University, USA, Dec. 1991. [5](#), [115](#)
1201. S. Owswitz and A. Sweetland. Factors affecting coding errors. Research Memorandum RM-4346-PR, The RAND Corporation, Apr. 1965. [19](#)
1202. S. C. Özbek. *Introducing Innovations into Open Source Projects*. PhD thesis, Freie Universität Berlin, Aug. 2010. [124](#)
1203. A. Ozment and S. E. Schechter. Milk or wine: Does software security improve with age? In *USENIX Security Symposium (2006)*, pages 93–104, July-Aug. 2006. [149](#), [150](#)
1204. P. Padfield. *Battleship*. Thistle Publishing, 2015. [2](#)
1205. R. Paleari, L. Martignoni, G. F. Roglia, and D. Bruschi. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, WOOT’09, pages 2–2, Aug. 2009. [138](#)
1206. N. Palix, J. Lawall, and G. Muller. Tracking code patterns over multiple software versions with Herodotus. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD’10, pages 169–180, Mar. 2010. [143](#)
1207. N. Palix, S. Saha, G. Thomas, C. Calvès, J. Lawall, and G. Muller. Faults in Linux: Ten years later. Technical Report RR-7357, Institut National de Recherche en Informatique et en Automatique, Aug. 2010. [143](#)
1208. J. Pallister, S. Hollis, and J. Bennett. Identifying compiler options to minimise energy consumption for embedded platforms. In *eprint arXiv:cs.PF/1303.6485*, Aug. 2013. [345](#)
1209. S. E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1999. [23](#)
1210. H.-Y. Pan, A. Chao, and W. Foissner. A nonparametric lower bound for the number of species shared by multiple communities. *Journal of Agricultural, Biological, and Environmental Statistics*, 14(4):452–468, Dec. 2009. [93](#)
1211. K. Pan. *Using Evolution Patterns to Find Duplicated Bugs*. PhD thesis, Department of Computer Science, University of California at Santa Cruz, Oct. 2006. [152](#)
1212. R. Parker and B. Grimm. Recognition of business and government expenditures for software as investment: Methodology and quantitative impacts, 1959–98. In *BEA Advisory Committee meeting*, May 2000. [55](#)
1213. A. Parkhomenko, A. Redkina, and O. Maslivets. Estimating hedonic price indexes for personal computers in Russia. MPRA Paper No. 5019, Higher School of Economics, Jan. 2007. [77](#)
1214. C. Parnin, C. Bird, and E. Murphy-Hill. Adoption and use of Java generics. *Empirical Software Engineering*, 18(6):1047–1089, Dec. 2013. [181](#)
1215. F. N. Parr. An alternative to the Rayleigh curve model for software development effort. *IEEE Transactions on Software Engineering*, SE-6(3):291–296, May 1980. [115](#)
1216. H. E. Pashler. *The Psychology of Attention*. The MIT Press, 1999. [22](#)
1217. L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wąsowski, and P. Borba. Coevolution of variability models and related artefacts: A case study of the Linux kernel. In *Proceedings of the 17th International Software Product Line Conference*, SPLC’13, pages 91–100, Apr. 2013. [87](#)
1218. A. Patel. Auditors’ belief revision: Recency effects of contrary and supporting audit evidence and source reliability. Technical Report 2001-1, University of South Pacific, Dept. of AFM/SSE, June 2001. [34](#)
1219. M. R. Patterson. *Antitrust Law in the New Economy :Google, Yelp, LIBOR, and the Control of Information*. Harvard University Press, 2017. [84](#), [91](#)
1220. F. M. Paulus, L. Rademacher, T. A. J. Schäfer, L. Müller-Pinzler, and S. Krach. Journal impact factor shapes scientists’ reward signal in the prospect of publication. *PLoS ONE*, 10(11):e0142537, Nov. 2015. [8](#)
1221. A. Pavese and C. Umiltà. Symbolic distance between numerosity and identity moulates stroop-like interference. *Journal of Experimental Psychology: Human Perception and Performance*, 24(5):1535–1545, 1998. [26](#)
1222. J. W. Payne, J. R. Bettman, and E. J. Bettman. *The Adaptive Decision Maker*. Cambridge University Press, 1993. [48](#), [49](#)
1223. G. Paz-y-Miño C, A. B. Bond, A. C. Kamil, and R. P. Balda. Pinyon jays use transitive inference to predict social dominance. *Nature*, 430:778–781, Aug. 2004. [41](#)
1224. R. D. Pea. Language-independent conceptual “bugs” in novice programming. *Journal of Educational Computing Research*, 2(1):25–36, Feb. 1986. [164](#)
1225. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000. [42](#)
1226. R. K. Pearson. The problem of disguised missing data. *ACM SIGKDD Explorations Newsletter*, 8(1):83–92, June 2006. [361](#)

1227. Y. Peers. *Econometric Advances in Diffusion Models*. PhD thesis, Erasmus University, Rotterdam, Dec. 2011. 78
1228. E. Pek, V. Klebanov, and R. Lämmel. Re-engineering software corpora for simplified adoption. ???, July 201? 347
1229. D. G. Pelli, C. W. Burns, B. Farrell, and D. C. Moore-Page. Feature detection and letter identification. *Vision Research*, 46(28):4646–4674, 2006. 25
1230. E. Peltonen, E. Lagerspetz, P. Nurmi, and S. Tarkoma. Energy modeling of system settings: A crowdsourced approach. In *IEEE International Conference on Pervasive Computing and Communications*, PerCom, pages 37–45, Mar. 2015. 348
1231. N. Pennington. Comprehension strategies in programming. In G. Olson, S. Shepard, and E. Soloway, editors, *Empirical Studies of Programmers: Second Workshop*, chapter 7, pages 100–113. Ablex Publishing, 1987. 168
1232. N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19(3):295–341, July 1987. 31, 183
1233. B. T. Pentland and H. H. Rueter. Organizational routines as grammars of action. *Administrative Science Quarterly*, 39(3):484–510, Sept. 1994. 95
1234. C. Perez. *Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages*. Edward Elgar Publishing, 2003. 2, 4
1235. D. E. Perry and W. M. Evangelist. An empirical study of software interface faults – An update. In *Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences, Vol II*, pages 113–126, Jan. 1987. 6, 141
1236. D. E. Perry, N. A. Staudenmayer, and L. G. Votta, Jr. Understanding and improving time usage in software development. In A. Fuggetta and A. L. Wolf, editors, *Trends in Software Process*, chapter 5, pages 111–135. John Wiley & Sons, Mar. 1995. 338
1237. D. E. Perry and C. S. Stieg. Software faults in evolving a large, real-time system: a case study. In *Proceedings of the 1993 European Software Engineering Conference*, pages 48–67, Sept. 1993. 141
1238. R. Perugupalli. Empirical assessment of architecture-based reliability of open-source software. Thesis (m.s.), Department of Computer Science and Electrical Engineering, West Virginia University, May 2004. 226
1239. H. Petroski. *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Cambridge University Press, 1994. 137
1240. C. Peukert. Switching costs and information technology: The case of IT outsourcing. ???, ???(??):???, Aug. 2010. 130
1241. A. Pewsey, M. Neuhäuser, and G. D. Ruxton. *Circular Statistics in R*. Oxford University Press, 2013. 323, 324, 325
1242. P. M. Pexman and M. J. Yap. Individual differences in semantic processing: Insights from the Calgary semantic decision project. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 44(7):1091–1112, Feb. 2018. 175
1243. S. A. Phatak, A. Lovitt, and J. B. Allen. Consonant confusions in white noise. *Journal of the Acoustic Society of America*, 124(2):1220–1233, Aug. 2008. 176
1244. C. Phillips. *Order and Structure*. PhD thesis, M.I.T., Aug. 1996. 29
1245. M. Phister, Jr. *Data Processing Technology and Economics*. Santa Monica Publishing Company and Digital Press, second edition, 1979. 5, 10, 94
1246. S. T. Piantadosi. Zipf's word frequency law in natural language: a critical review and future directions. *Psychonomic Bulletin & Review*, 21(5):1112–1130, Oct. 2014. 184
1247. S. T. Piantadosi. A rational analysis of the approximate number system. *Psychonomic Bulletin & Review*, 23(3):877–886, June 2016. 44
1248. R. Pieters and L. Warlop. Visual attention during brand choice: The impact of time pressure and task motivation. *International Journal of Research in Marketing*, 16:1–16, 1999. 24
1249. D. J. Pigott and B. M. Axtens. Online historical encyclopedia of programming languages. <http://hopl.info/>, 2015. 102
1250. R. S. Pindyck. Investments of uncertain cost. *Journal of Financial Economics*, 34(1):53–76, Aug. 1993. 60
1251. J. Pipitone. Software quality in climate modelling. Thesis (m.s.), Department of Computer Science, University of Toronto, 2010. 145
1252. A. M. Pires and C. ao Amado. Interval estimators for a binomial proportion: Comparison of twenty methods. *REVSTAT-Statistical Journal*, 6(2):165–197, June 2008. 246
1253. D. J. Pittenger. Measuring the MBTI ... And coming up short. *Journal of Career Planning and Employment*, 54(1):48–52, Nov. 1993. 47
1254. PK. How many developers are there in America, and where do they live? website, Apr. 2019. <https://dqydj.com/number-of-developers-in-america-and-per-state/>. 92
1255. A. Pluchino, A. Rapisarda, and C. Garofalo. The Peter principle revisited: A computational study. In *eprint arXiv:physics.soc-ph/0907.0455v3*, Oct. 2009. 64
1256. T. Plum. *Reliable data structures in C*. Plum Hall, 1985. 166
1257. T. Plum. *C Programming guidelines*. Plum Hall, 1989. 166
1258. I. P. L. Png. On the reliability of software piracy statistics. *Electronic Commerce Research and Applications*, 9(5):365–373, Sept.-Oct. 2010. 79
1259. C. Poivey, J. L. Barth, K. A. LaBel, G. Gee, and H. Safran. In-flight observations of long-term single-event effect (SEE) performance on Orbview-2 solid state recorders (SSR). In *2003 IEEE Radiation Effects Data Workshop*, page ???, July 2003. 203
1260. R. Pollack. How to believe a machine-checked proof. In G. Sambin and J. M. Smith, editors, *Twenty Five Years of Constructive Type Theory*, chapter 11, pages 205–220. Oxford University Press, Oct. 1998. 138
1261. A. Pollatsek, E. D. Reichle, and K. Rayner. Tests of the E-Z reader model: Exploring the interface between cognition and eye-movement control. *Cognitive Psychology*, 52(1):1–56, Feb. 2006. 25
1262. D. Pope and U. Simonsohn. Round numbers as goals: Evidence from baseball, SAT takers, and the lab. *Psychological Science*, 22(1):71–79, Jan. 2011. 45
1263. K. R. Popper. *Conjectures and Refutations*. Routledge, 1969. 21
1264. A. Porter, H. Siy, A. Mockus, and L. Votta. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering Methodology*, 7(1):41–79, Jan. 1998. 159, 281, 339
1265. M. E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. First Free Press, 1985. 76
1266. M. E. Porter. The five competitive forces that shape strategy. *Harvard Business Review*, 86(1):78–93, Jan. 2008. 56, 90
1267. R. D. Portugal and B. F. Svaiter. Weber-Fechner law and the optimality of the logarithmic scale. *Minds & Machines*, 21(1):73–81, Feb. 2011. 53
1268. A. S. Posamentier and I. Lehmann. *Magnificent mistakes in mathematics*. Prometheus books, 2013. 138
1269. D. E. Post and R. P. Kendall. Software project management and quality engineering practices for complex, coupled multi-physics, massively parallel computational simulations: Lessons learned from ASCI. Report LA-UR-03-1274 Rev. 2, Los Alamos National Laboratory, Mar. 2004. 101
1270. A. Potanin, M. Damitio, and J. Noble. Are your incoming aliases really necessary? Counting the cost of object ownership. In *Proceedings of the 2013 International Conference on Software*, ICSE'13, pages 742–751, May 2013. 248
1271. E. M. Pothos and N. Chater. Rational categories. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, pages 848–853, 1998. 36
1272. M. C. Potter, A. Moryadas, I. Abrams, and A. Noel. Word perception and misperception in context. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 19(1):3–22, 1993. 176
1273. J. Potts, J. Hartley, L. Montgomery, C. Neylon, and E. Rennie. A journal is a club: A new economic model for scholarly publishing. SSRN Working Paper n. 2763975, Apr. 2016. 9
1274. A. L. Powell. *Right on Time: Measuring, Modelling and Managing Time-Constrained Software Development*. PhD thesis, Department of Computer Science, University of York, Aug. 2001. 111, 312, 313
1275. D. A. Powner and K. A. Rhodes. Business systems modernization: IRS needs to complete recent efforts to develop policies and procedures to guide requirements development and management. Technical Report GAO-06-310, United States Government Accountability Office, Mar. 2006. 128

1276. M. Pradel. *Program Analyses for Automatic and Precise Error Detection*. PhD thesis, ETH Zurich, 2012. 147, 148
1277. L. Prechelt. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really? Technical Report iratr-1999-18, Universität Karlsruhe, 1999. 7, 52, 200
1278. L. Prechelt. Plat\_Forms 2007: The web development platform comparison-evaluation and results. Technical Report B-07-10, Institut für Informatik, Freie Universität Berlin, June 2007. 122, 192
1279. L. Prechelt, D. Graziotin, and D. M. Fernández. On the status and future of peer review in software engineering. In *eprint arXiv:cs.SE/1706.07196*, June 2017. 8
1280. L. Prechelt and W. F. Tichy. A controlled experiment measuring the effect of procedure argument type checking on programmer productivity. *IEEE Transactions on Software Engineering*, 24(4):302–312, Apr. 1998. 167
1281. L. Prechelt, F. Zieris, and H. Schmeisky. Difficulty factors of obtaining access for empirical studies in industry. In *Proceedings of the Third International Workshop on Conducting Empirical Studies in Industry*, CESI'15, pages 19–25, May 2015. 6
1282. L. S. Premo and S. L. Kuhn. Modeling effects of local population extinctions on cultural change and diversity in the paleolithic. *PLoS ONE*, 5(12):e15582, Dec. 2010. 68, 205
1283. C. C. Presson and D. R. Montello. Updating after rotational and translational body movements: coordinate structure of perspective space. *Perception*, 23:1447–1455, 1994. 18
1284. D. Pritchard. Frequency distribution of error messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2015, pages 1–8, Oct. 2015. 153
1285. V. Propp. *Morphology of the Folktale*. University of Texas Press, second edition, 1968. English translation by Laurence Scott. 182
1286. J. Prümper, D. Zapf, F. C. Brodbeck, and M. Frese. Some surprising differences between novice and expert errors in computerized office work. *Behaviour & Information Technology*, 11(6):319–328, 1992. 137
1287. D. Pukhkaiev. Energy-efficient benchmarking for energy-efficient software. Thesis (m.s.), Technische Universität, Dresden, Dec. 2015. 342
1288. R. Purushothaman and D. E. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, June 2005. 154, 155
1289. L. H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, SE-4(4):345–361, July 1978. 115
1290. L. H. Putnam and W. Myers. *Measures for Excellence: Reliable software on time, within budget*. Prentice-Hall, Inc, 1992. 208
1291. PwC. Converging forces are building that could re-shape the entire industry. Global 100 software leaders, PwC Technology Institute, May 2013. 79
1292. PwC. The growing importance of apps and services. Global 100 software leaders, PwC Technology Institute, Mar. 2014. 79
1293. PwC. Digital intelligence conquers the world below and the cloud above. Global 100 software leaders, PwC Technology Institute, 2016. 79
1294. PwC. IPO review full-year and Q4 2015. Global technology, PwC Technology Institute, Feb. 2016. 83
1295. Z. Pylyshyn. Is vision continuous with cognition? The case for cognitive impenetrability of visual perception. *Behavioral and Brain Sciences*, 22(3):341–423, 1999. 23
1296. X. Qu. *Configuration aware prioritization for regression testing*. PhD thesis, The Graduate College at the University of Nebraska, Apr. 2010. 161
1297. S. Qualline. *C Elements of Style*. M&T Books, 1992. 166
1298. R. Queiroz, L. Passos, M. T. Valente, C. Hunsen, S. Apel, and K. Czarnecki. The shape of feature code: an analysis of twenty C-preprocessor-based systems. *Journal on Software and Systems Modeling*, 16(1):77–96, Feb. 2017. 297, 298
1299. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. ISBN 3-900051-07-0. 2, 367, 368
1300. R Core Team. R language definition. Technical Report 3.3.1, R Foundation for Statistical Computing, June 2016. 367
1301. H. Rabinowitz and C. Schaap. *Portable C*. Prentice-Hall, Inc, 1990. 166
1302. J. J. Rachlinski, A. J. Wistrich, and C. Guthrie. Can judges make reliable numeric judgments? Distorted damages and skewed sentences. *Indiana Law Journal*, 90(2):695–739, Apr. 2015. 44
1303. J. W. Radatz. Analysis of IV & V data. Technical Report RADC-TR-81-145, Rome Air Development Center, Griffiss Air Force Base, June 1981. 151
1304. G. Radden and R. Dirven. *Cognitive English Grammar*. John Benjamins Publishing Company, 2007. 163
1305. D. Raffo, J. Settle, and W. Harrison. Investigating financial measures for planning software IV&V. Technical Report TR-99-05, Portland State University, 1999. 58
1306. C. Ragkhitwetsagul, J. Krinke, and D. Clark. A comparison of code similarity analysers. *Empirical Software Engineering*, 23(4):2464–2519, Aug. 2018. 179
1307. F. Rahman, C. Bird, and P. Devanbu. Clones: What is that smell? In *Proceedings of the 7th International Workshop on Mining Software Repositories*, MSR'10, pages 72–81, May 2010. 6
1308. J. Ranade and A. Nash. *The Elements of C Programming Style*. McGraw-Hill, Inc, 1992. 166
1309. R. Ratcliff, G. McKoon, and P. Gomez. A diffusion model account of the lexical decision task. *Psychological Review*, 111(1):159–182, Jan. 2004. 18
1310. R. Ratcliff, P. L. Smith, S. D. Brown, and G. McKoon. Diffusion decision model: Current issues and history. *Trends in Cognitive Sciences*, 20(4):260–281, Apr. 2016. 18
1311. B. Ray. *Analysis of Cross-System Porting and Porting Errors in Software Projects*. PhD thesis, University of Texas at Austin, Aug. 2013. 87, 88, 143
1312. B. Ray, M. Wilcox, and C. Voskoglou. Developer economics | State of the developer nation q3 2015. State of the Nation 9th edition, VisionMobile, July-Sept. 2015. 103
1313. K. Rayner. Eye movements and attention in reading, scene perception, and visual search. *The Quarterly Journal of Experimental Psychology*, 62(8):1457–1506, 2009. 24
1314. K. Rayner. Eye movements in reading: Models and data. *Journal of Eye Movement Research*, 2(5):1–10, Apr. 2009. 24
1315. N. M. Razali and Y. B. Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33, 2011. 242
1316. J. Reason. *Human Error*. Cambridge University Press, 1990. 137, 151
1317. A. S. Reber and S. M. Kassin. On the relationship between implicit and explicit modes in the learning of a complex rule structure. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):492–502, 1980. 32
1318. J. L. Recon. Building skyscrapers, and spending on major projects. [https://github.com/jlrecon/open\\_nintil](https://github.com/jlrecon/open_nintil), Oct. 2018. <https://nintil.com/2018/10/07/building-skyscrapers-and-spending-on-major-projects/>. 110
1319. B. Regnell, M. Höst, J. N. och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, Apr. 2001. 49, 128
1320. E. D. Reichle, T. Warren, and K. McConnell. Using E-Z reader to model the effects of higher-level language processing on eye movements during reading. *Psychonomic Bulletin & Review*, 16(1):1–21, Feb. 2009. 25
1321. R. J. Reid. The Reid list of the first course language for computer science majors. <http://www.csee.wvu.edu/~vanscoy/reid.htm>, Aug. 2002. 104
1322. J. Reimer. Computer smartphone and tablet marketshare: 1975-2012. website, Dec. 2012. <http://jeremyreimer.com/m-item.lsp?i=137>. 3, 82
1323. G. A. Reis III. *Software Modulated Fault Tolerance*. PhD thesis, Department of Electrical Engineering, Princeton University, June 2008. 156

1324. G. Remillard. Implicit learning of second-, third-, and fourth-order adjacent and nonadjacent sequential dependencies. *The Quarterly Journal of Experimental Psychology*, 61(3):400–424, Apr. 2008. 26
1325. R. W. Remington, H. W. H. Yuen, and H. Pashler. With practice, keyboard shortcuts become faster than menu selection: A crossover interaction. *Journal of Experimental Psychology: Applied*, 22(1):95–106, 2016. 36
1326. Research Councils UK. RCUK policy on open access and supporting guidance. Technical Report ???, RCUK, Apr. 2013. 8
1327. G. Richards, C. Hammer, B. Burg, and J. Vitek. The eval that men do: A large-scale study of the use of eval in JavaScript applications. In *Proceedings of the 25th European conference on Object-oriented programming*, ECOOP’11, pages 52–78, July 2011. 179
1328. G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI’10, pages 1–12, June 2010. 179
1329. R. Richardson. 2008 CSI computer crime & security survey. Technical report, Computer Security Institute, Aug. 2008. 143
1330. D. F. Rico. Short history of software methods. <http://davidfrico.com/rico04e.pdf>, July 2004. 123
1331. R. Riesen, K. Ferreira, J. Stearley, R. Oldfield, J. H. Laros III, K. Pedretti, and R. Brightwell. Redundant computing for exascale systems. Technical Report SAND2010-8709, Sandia National Laboratories, Dec. 2010. 156
1332. M. Rigger, S. Marr, B. Adams, and H. Mössenböck. Understanding GCC builtins to develop better tools. In *eprint arXiv:cs.PL/1907.00863*, July 2019. 104
1333. M. Rigger, S. Marr, S. Kell, D. Leopoldseder, and H. Mössenböck. An analysis of x86-64 inline assembly in C programs. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE’18, pages 84–99, Mar. 2018. 183
1334. M. Rinard, C. Cadar, and H. H. Nguyen. Exploring the acceptability envelope. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA’05, pages 21–30, Oct. 2005. 137
1335. M. Ringelmann. Recherches sur les moteurs animés: Travail de l’homme. *Annales de l’Institut National Agronomique*, 2(XII):1–40, 1913. 72
1336. R. Robbes, D. Röthlisberger, and É. Tanter. Object-oriented software extensions in practice. *Empirical Software Engineering*, 20(3):745–782, June 2015. 185, 186
1337. M. J. Roberts, D. J. Gilmore, and D. J. Wood. Individual differences and strategy selection in reasoning. *British Journal of Psychology*, 88:473–492, 1997. 40
1338. S. Roberts and J. Winters. Linguistic diversity and traffic accidents: Lessons from statistical studies of cultural traits. *PLoS ONE*, 8(8):e70902, Aug. 2013. 245
1339. D. E. Robinson. Fashions in shaving and trimming of the beard: The men of the Illustrated London News, 1842–1972. *American Journal of Sociology*, 81(5):1133–1141, Mar. 1976. 7
1340. G. Robles and J. M. González-Barahona. A comprehensive study of software forks: Dates, reasons and outcomes. In *The 8th International Conference on Open Source Systems*, OSS 2012, pages 1–14, Sept. 2012. 86, 87
1341. G. Robles, I. Herráiz, D. M. Germán, and D. Izquierdo-Cortázar. Modification and developer metrics at the function level: Metrics for the study of the evolution of a software project. In *3rd International Workshop on Emerging Trends in Software Metrics*, WETSoM, pages 49–55, June 2012. 189, 190, 191
1342. G. Robles, L. A. Reina, A. Serebrenik, B. Vasilescu, and J. M. González-Barahona. FLOSS 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR’14, pages 396–399, May 2014. 196, 354
1343. E. Rodrigues, Jr. and R. Terra. How do developers use dynamic features? The case of Ruby. *Computer Languages, Systems & Structures*, 53:73–89, Sept. 2018. 184
1344. W. H. Roetzheim. When the software becomes a nightmare: Dealing with failed projects. *Business Law Today*, 13(6):42–48, July-Aug. 2004. 125
1345. R. D. Rogers and S. Monsell. Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, 124(2):207–231, 1995. 22
1346. E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8(3):382–439, July 1976. 36
1347. L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall PTR, July 2004. 62
1348. M. Rosenfelder. *The Language Construction Kit*. Yonagu Books, 2010. 102
1349. A. Ross. *No-Collar: The Humane Workplace and its Hidden Costs*. Temple University Press, 2003. 67
1350. B. H. Ross and G. L. Murphy. Food for thought: Cross-classification and category organization in a complex real-world domain. *Cognitive Psychology*, 38(4):495–552, June 1999. 334
1351. L. Ross, M. R. Lepper, and M. Hubbard. Perseverance in self-perception and social perception: Biased attributional processes in the debelieving paradigm. *Journal of Personality and Social Psychology*, 32(5):880–892, 1975. 34
1352. B. Rossi, B. Russo, and G. Succi. Path dependent stochastic models to detect planned and actual technology use: A case study of openoffice. *Information & Software Technology*, 53(11):1209–1226, Nov. 2011. 92
1353. J. Rost and R. L. Glass. *The Dark Side of Software Engineering: Evil on Computing Projects*. John Wiley & Sons, Inc, 2011. 111, 127
1354. V. Rothberg, N. Dintzner, A. Ziegler, and D. Lohmann. Feature models in Linux—from symbols to semantics. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS’16, pages 65–72, Jan. 2016. 87
1355. B. F. Roukema. A first-digit anomaly in the 2009 Iranian presidential election. In *eprint arXiv:stat.AP/0906.2789*, June 2013. 366
1356. M. M. Roy, N. J. S. Christenfeld, and C. R. M. McKenzie. Underestimating the duration of future events: Memory incorrectly used or memory bias? *Psychological Bulletin*, 131(5):738–756, 2005. 50
1357. W. W. Royce. Managing the development of large software systems. In *Proceedings IEEE WESON*, pages 1–9, Aug. 1970. 122
1358. P. Royston, D. G. Altman, and W. Sauerbrei. Dichotomizing continuous predictors in multiple regression: a bad idea. *Statistics in Medicine*, 25(1):127–141, Jan. 2006. 282
1359. D. C. Rubin, S. Hinton, and A. Wenzel. The precise time course of retention. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 25(5):1161–1176, Sept. 1999. 31, 32
1360. D. C. Rubin and A. E. Wenzel. One hundred years of forgetting: A quantitative description of retention. *Psychological Review*, 103(4):734–760, 1996. 31
1361. C. Rubio-González and B. Libit. Expect the unexpected: Error code mismatches between documentation and the real world. In *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE’10, pages 73–80, June 2010. 155
1362. C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC’13, Nov. 2013. 139
1363. J. Ruohonen and V. Leppänen. How PHP releases are adopted in the wild? In *eprint arXiv:cs.SE/1710.05570*, Oct. 2017. 88
1364. A. L. Russell. ‘rough consensus and running code’ and the internet-OSI standards war. *IEEE Annals of the History of Computing*, 28(3):48–61, July-Sept. 2006. 123
1365. R. Sabherwal, A. Jeyaraj, and C. Chowa. Information systems success: Individual and organizational determinants. *Management Science*, 52(12):1849–1864, Dec. 2006. 242
1366. R. Saborido, V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol. On the impact of sampling frequency on software energy measurements. *PeerJ PrePrints*, 3:e1219, July 2015. 349
1367. H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, 11(1):3–11, Jan. 1968. 7
1368. M. Sadat, A. B. Bener, and A. V. Miranskyy. Rediscovery datasets: Connecting duplicate reports. In *eprint arXiv:cs.SE/1703.06337v1*, Mar. 2017. 141, 149, 151

1369. M. Sadinle. On the performance of dual system estimators of population size: A simulation study. Documentos de CERAC No. 13, Centro de Recursos para el Análisis de Conflictos, Bogotá, Columbia, Dec. 2008. [93](#)
1370. S. K. Sahoo, J. Criswell, and V. Adve. An empirical study of reported bugs in server software with implications for automated bug diagnosis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, ICSE'10, pages 485–494, May 2010. [141](#)
1371. J. Sajaniemi and R. N. Prieto. Roles of variables in experts' programming knowledge. In *17th Workshop of the Psychology of Programming Interest Group*, pages 145–159, June 2005. [185](#)
1372. P. H. Salus. *A Quarter Century of UNIX*. Addison-Wesley, 1994. [104](#)
1373. P. H. Salus. Duelling UNIXes and the UNIX wars. *;login;*, 40(2):66–68, Apr. 2015. [104](#)
1374. A. Sampson, W. Dietl, E. Fortuna, and D. Gnanapragasam. EnerJ: Approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, PLDI'11, pages 164–174, June 2011. [348](#)
1375. D. M. Sanbonmatsu, S. S. Posavac, A. A. Behrends, S. M. Moore, and B. N. Uchino. Why a confirmation strategy dominates psychological science. *PLoS ONE*, page e0138197, Sept. 2015. [21](#)
1376. D. Sarkar. *Lattice Multivariate Data Visualization with R*. Springer Science+Business Media, 2008. [203](#)
1377. M. Savić, M. Ivanović, Z. Budimac, and M. Radovanović. Do students' programming skills depend on programming language? In *American Institute of Physics Conference Proceedings*, page 240006, June 2016. [103](#)
1378. S. R. Schach, T. O. S. Adeshiyen, D. Balasubramanian, G. Madl, E. P. Osses, S. Singh, K. Suwanmongkol, M. Xie, and D. G. Feitelson. Common coupling and pointer variables, with application to a Linux case study. *Software Quality Journal*, 15(1):99–113, Mar. 2006. [159](#)
1379. S. R. Schach, B. Jin, L. Yu, G. Z. Heller, and J. Offutt. Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering*, 8(4):351–363, Dec. 2003. [134, 304, 305](#)
1380. J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. In *Proceedings of the VLDB Endowment*, pages 460–471, Sept. 2010. [354, 356](#)
1381. K. W. Schaie. *Developmental Influences on Adult Intelligence: The Seattle Longitudinal Study*. Oxford University Press, second edition, 2013. [53](#)
1382. R. R. Schaller. *Technological Innovation in the Semiconductor Industry: A Case Study of the International Technology Roadmap for Semiconductors (ITRS)*. PhD thesis, George Mason University, 2004. [84](#)
1383. M. Schief. *Business Models in the Software Industry: The Impact on Firm and M&A Performance*. Springer Gabler, Apr. 2014. [76](#)
1384. F. L. Schmidt, I.-S. Oh, and J. A. Shaffer. The validity and utility of selection methods in personnel psychology: Practical and theoretical implications of 100 years of research findings. Working paper, SSRN ???, Aug. 2016. [17](#)
1385. E. Schneider, M. Maruyama, S. Dehaene, and M. Sigman. Eye gaze reveals a fast, parallel extraction of the syntax of arithmetic formulas. *Cognition*, 125(3):475–490, Dec. 2012. [25](#)
1386. S. Schneider. The dirty little secret of software pricing. website, 2012. [http://www.rti.com/whitepapers/Dirty\\_Little\\_Secret.pdf](http://www.rti.com/whitepapers/Dirty_Little_Secret.pdf). [76](#)
1387. J. Schock, M. J. Cortese, M. M. Khanna, and S. Toppi. Age of acquisition estimates for 3,000 disyllabic words. *Behavior and Research Methods*, 44(4):971–977, Dec. 2012. [175](#)
1388. A. Scholey and L. Owen. Effects of chocolate on cognitive function and mood: a systematic review. *Nutrition Reviews*, 71(10):665–681, Apr. 2013. [52](#)
1389. R. Schöne, D. Hackenberg, and D. Molka. Memory performance at reduced CPU clock speeds: An analysis of current x86\_64 processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, Oct. 2012. [202, 351](#)
1390. M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, 2001. [66](#)
1391. L. J. Schooler and R. Hertwig. How forgetting aids heuristic inference. *Psychological Review*, 112(3):610–628, 2005. [31](#)
1392. E. R. Schotter, B. Angele, and K. Rayner. Parafoveal processing in reading. *Attention, Perception & Psychophysics*, 74(1):5–35, Jan. 2012. [25](#)
1393. J.-P. Schraepler and G. G. Wagner. Identification of faked interviews in surveys by means of Benford's law?: An analysis by means of genuine fakes in the raw data of SOEP. Technical report, Technische Universität Berlin, Aug. 2004. [366](#)
1394. A. Schulman, M. Pietrek, and D. Maxey. *Undocumented Windows: A Programmers Guide to Reserved Microsoft Windows API Functions*. Addison-Wesley, July 1992. [106](#)
1395. M.-A. Schulz, B. Schmalbach, P. Brugger, and K. Witt. Analysing humanly generated random number sequences: A pattern-based approach. *PLoS ONE*, 7(7):e41531, July 2012. [366](#)
1396. P. Schuurman, E. Berghout, and P. Powell. Benefits are from Venus, costs are from Mars. CITER WP/010/PSEBPP, University of Groningen Centre for IT Economics Research, June 2008. [109](#)
1397. E. S. Schwartz and C. Zozaya-Gorostiza. Investment under uncertainty in information technology: Acquisition and development projects. *Management Science*, 49(1):57–70, Jan. 2003. [60](#)
1398. C. Scott. Numbers every programmer should know. website, Oct. 2016. [https://github.com/colin-scott/interactive\\_latencies](https://github.com/colin-scott/interactive_latencies). [208, 209](#)
1399. C. F. Scott, P. Cole, R. B. Hesse, and P. R. Malone. UNITED STATES OF AMERICA, et al., v. ORACLE CORPORATION. Plaintiff's post-trial brief CASE NO. C 04-0807 VRW, UNITED STATES DISTRICT COURT NORTHERN DISTRICT OF CALIFORNIA SAN FRANCISCO DIVISION, July 2004. [91](#)
1400. P. D. Scott and M. Fasli. Benford's law: An empirical investigation and a novel explanation. CSM Technical Report 349, Department of Computer Science, University of Essex, Aug. 2001. [366](#)
1401. S. Scribner. Modes of thinking and ways of speaking: culture and logic reconsidered. In P. N. Johnson-Laird and P. C. Wason, editors, *Thinking: Readings in Cognitive Science*, chapter 29, pages 483–500. Cambridge University Press, 1977. [39](#)
1402. R. C. Seacord. *The CERT C Secure Coding Standard*. Addison-Wesley, 2009. [166](#)
1403. R. C. Seamans, Jr. *Aiming at Targets: The Autobiography of Robert C. Seamans, Jr.* NASA History Office, 1996. [110](#)
1404. SEC. The world's largest hedge fund is a fraud. SEC MADOFF EXHIBITS-04451, Nov. 2005. November 7, 2005 Submission to the SEC, Madoff Investment Securities, LLC. [366](#)
1405. P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, Feb. 2010. [352](#)
1406. R. W. Selby, Jr., V. R. Basili, and F. T. Baker. CLEANROOM software development: An empirical evaluation. Technical Report TR-1415, Department of Computer Science, University of Maryland, Feb. 1985. [118](#)
1407. L. L. Selwyn. *Economies of Scale in Computer Use: Initial Tests and Implications for the Computer Utility*. PhD thesis, Alfred P. Sloan School of Management, June 1969. [94](#)
1408. J. A. Sexton. Detecting errors in software using a parameter checker: An analysis. Thesis (m.s.), Rochester Institute of Technology, Apr. 1989. [152](#)
1409. N. Shadbolt. Shadbolt review of computer sciences degree accreditation and graduate employability. Technical Report IND/16/5, Department for Business, Innovation & Skills, UK, Apr. 2016. [66, 340](#)
1410. T. M. Shaft and I. Vessey. The relevance of application domain knowledge: Characterizing the computer program comprehension process. *Journal of Management Information Systems*, 15(1):51–78, 1998. [132, 168](#)
1411. C. R. Shaliz. g, a statistical myth. Three-Toed Sloth: blog, Oct. 2007. <http://bactra.org/weblog/523.html>. [47](#)
1412. J. Shallit. Randomized algorithms in "primitive" cultures or what is the oracle complexity of a dead chicken? *ACM SIGACT News*, 23(4):77–80, Sept.–Dec. 1992. [168](#)
1413. C. Shaoul, R. H. Baayen, and C. F. Westbury. N-gram probability effects in a cloze task. *The Mental Lexicon*, 9(3):437–472, 2014. [176](#)

1414. C. Shapiro and H. R. Varian. The art of standards wars. *California Management Review*, 41(2):8–32, Jan. 1999. 73
1415. W. F. Sharpe. *The Economics of Computers*. Columbia University Press, 1969. 94
1416. O. Shatnawi. Measuring commercial software operational reliability: an interdisciplinary modelling approach. *Eksplotacija i Niezawodnosć – Maintenance and Reliability*, 16(4):585–594, 2014. 145
1417. D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Pipana, Y. Shan, and B. Towles. Millisecond-scale molecular dynamics simulations on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC’09, page 39, Nov. 2009. 94
1418. B. R. Shear and B. D. Zumbo. False positives in multiple regression: Unanticipated consequences of measurement error in the predictor variables. *Educational and Psychological Measurement*, 73(5):733–756, Oct. 2013. 266
1419. D. Shefer. Pricing for software product managers. ???, 2005. 77
1420. B. A. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, Mar. 1981. 6
1421. S. Shekhar, M. Dietz, and D. S. Wallach. AdSplit: Separating smartphone advertising from applications. In *Proceedings of the 21st USENIX conference on Security symposium*, Security’12, page 28, Aug. 2012. 80
1422. T.-J. Shen, A. Chao, and C.-F. Lin. Predicting the number of new species in further taxonomic sampling. *Ecology*, 84(3):798–804, Mar. 2003. 93
1423. V. Y. Shen, S. D. Conte, and H. E. Dunsmore. Software science revisited: A critical analysis of the theory and its empirical support. Technical Report CSD-TR 375, Purdue University, Jan. 1981. 178
1424. A. Shenhav, S. Musslick, F. Lieder, W. Kool, T. L. Griffiths, J. D. Cohen, and M. M. Botvinick. Toward a rational and mechanistic account of mental effort. *Annual Review of Neuroscience*, 40:99–124, July 2017. 22
1425. R. N. Shepard, C. I. Hovland, and H. M. Jenkins. Learning and memorization of classifications. *Psychological Monographs: General and Applied*, 75(15):1–39, 1961. 37, 38
1426. R. N. Shepard and J. Metzler. Mental rotation of three-dimensional objects. *Science*, 171:701–703, Feb. 1971. 18
1427. S. B. Sheppard and E. Kruesi. The effects of the symbology and spatial arrangement of software specifications in a coding task. Technical Report TR-81-388200-3, Information Systems Programs, General Electric, Feb. 1981. 152
1428. M. Shepperd, C. Mair, and M. Jørgensen. An experimental evaluation of a de-biasing intervention for professional software developers. In *eprint arXiv:cs.SE/1804.03919*, Apr. 2018. 113
1429. L. Shi, H. Zhong, T. Xie, and M. Li. An empirical study on evolution of API documentation. In *Proceedings of the 14th international conference on Fundamental approaches to software engineering*, FASE’11/ETAPS’11, pages 416–431, Apr. 2011. 107
1430. E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. ichi Matsumoto. Predicting re-opened bugs: A case study on the Eclipse project. In *17th Working Conference on Reverse Engineering*, WCRE’10, pages 249–258, Oct. 2010. 330, 331
1431. E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan. Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’10, pages 1–4, Sept. 2010. 153
1432. M. Shimasaki, S. Fukaya, K. Ikeda, and T. Kiyono. An analysis of Pascal programs in compiler writing. *Software—Practice and Experience*, 10(2):149–157, Feb. 1980. 117, 118
1433. A. L. Shimpi and B. Klug. They’re (almost) all dirty: The state of cheating in Android benchmarks. website, Oct. 2013. <http://www.anandtech.com/show/7384/state-of-cheating-in-android-benchmarks>. 346
1434. T. C. Shrum. Calibration and validation of the checkpoint model to the air force electronic systems center software database. Thesis (m.s.), Graduate School of Logistics and Acquisition Management or the Air Force Institute of Technology, Air University, Sept. 1997. 5
1435. A. Shtub, N. Levin, and S. Globerson. Learning and forgetting industrial skills: An experimental model. *The International Journal of Human Factors in Manufacturing*, 3(3):293–305, July 1993. 34
1436. O. Shy. *How to Price: A Guide to Pricing Techniques and Yield Management*. Cambridge University Press, 2008. 76
1437. R. M. Siegfried, J. P. Siegfried, and G. Alexandro. A longitudinal analysis of the Reid list of first programming languages. *Information Systems Education Journal*, 14(6):47–54, Nov. 2016. 104
1438. N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. *Information and Software Technology*, 55(3):491–507, Mar. 2013. 183
1439. R. Silberzahn, E. L. Uhlmann, D. P. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahník, F. Bai, C. Bannard, E. Bonnier, R. Carlsson, F. Cheung, G. Christensen, R. Clay, M. A. Craig, A. D. Rosa, L. Dam, M. H. Evans, I. F. Cervantes, N. Fong, M. Gamez-Djokic, A. Glenz, S. Gordon-McKeon, T. J. Heaton, K. Hederos, M. Heene, A. J. H. Mohr, F. Högden, K. Hui, M. Johannesson, J. Kalodimos, E. Kaszubowski, D. M. Kennedy, R. Lei, T. A. Lindsay, S. Liverani, C. R. Madan, D. Molden, E. Molleman, R. D. Morey, L. B. Mulder, B. R. Nijstad, N. G. Pope, B. Pope, J. M. Prenoveau, F. Rink, E. Robusto, H. Roderique, A. Sandberg, E. Schlüter, F. D. Schönbrodt, M. F. Sherman, S. A. Sommer, K. Sotak, S. Spain, C. Spörlein, T. Stafford, L. Stefanutti, S. Tauber, J. Ullrich, M. Vianello, E.-J. Wagenaemakers, M. Witkowiak, S. Yoon, and B. A. Nosek. Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*, 1(3):337–356, Apr. 2018. 231
1440. T. Simcoe. Standard setting committees: Consensus governance for shared technology platforms. *American Economic Review*, 102(1):305–336, Feb. 2013. 73
1441. T. Simcoe. Modularity and the evolution of the internet. In A. Goldfarb, S. M. Greenstein, and C. E. Tucker, editors, *Economic Analysis of the Digital Economy*, chapter 1, pages 21–47. University of Chicago Press, May 2015. 122
1442. T. S. Simcoe and D. M. Wagstaff. Status, quality and attention: What’s in a (missing) name? *Management Science*, 57(2):274–290, Sept. 2011. 67
1443. K. M. Simmons and D. Sutter. False alarms, tornado warnings, and tornado casualties. *Weather, Climate, and Society*, 1(1):38–53, Oct. 2009. 211
1444. H. A. Simon. *Models of Bounded Rationality: Behavioral Economics and Business Organization*. The MIT Press, 1982. 48
1445. H. A. Simon. Making management decisions: the role of intuition and emotion. *The Academy of Management Executive (1987-1989)*, 1(1):57–64, Feb. 1987. 34
1446. I. Simonson. Choice based on reasons: The case of attraction and compromise effects. *Journal of Consumer Research*, 16:158–173, Sept. 1989. 49
1447. I. C. Simpson, P. Mousikou, J. M. Montoya, and S. Defior. A letter visual-similarity matrix for Latin-based alphabets. *Behavior and Research Methods*, 45(2):431–439, June 2013. 19
1448. J. Singer, M. Luján, and I. Watson. Meaningful type names as a basis for object lifetime prediction. In *Proceedings of the 2008 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, OOPSLA’08, page ???, Apr. 2008. 182
1449. P. V. Singh and C. Phelps. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, 24(3):539–560, Nov. 2013. 62
1450. D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, E. F. Koren, and M. Vokáč. Conducting realistic experiments in software engineering. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, ISESE’02, pages 17–26, Oct. 2002. 340
1451. D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. Technical Report 2004-4, SIMULA Research Laboratory, 2004. 338

1452. D. Skau and R. Kosara. Arcs, angles, or areas: Individual data encodings in pie and donut charts. In *Eurographics Conference on Visualization*, EuroVis'16, pages 121–130, June 2016. [202](#)
1453. J. Skelley. Open source tactics: Bargaining power for strategic litigation. *Chicago-Kent Journal of Intellectual Property*, 16(1), 2016. [64](#)
1454. I. Skoulis. Analysis of schema evolution for databases in open-source software. Thesis (m.s.), University of Ioannina, Greece, Sept. 2013. [188](#), [189](#)
1455. G. Slade. *Made to Break*. Harvard University Press, 2007. [4](#)
1456. S. A. Slaughter, S. Ang, and W. F. Boh. Firm-specific human capital and compensation-organizational tenure profiles: An archival analysis of salary data for IT professionals. *Human Resource Management*, 46(3):373–394, 2007. [65](#)
1457. S. A. Sloman. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119(1):3–22, 1996. [39](#)
1458. S. A. Sloman. Categorical inference is not a tree: The myth of inheritance hierarchies. *Cognitive Psychology*, 35(1):1–33, Feb. 1998. [37](#)
1459. S. A. Sloman, M. C. Harrison, and B. C. Malt. Recent exposure affects artifact naming. *Memory & Cognition*, 30(5):687–695, 2002. [95](#)
1460. S. A. Sloman and D. Lagnado. Causality in thought. *Annual Review of Psychology*, 66:223–247, 2015. [42](#)
1461. S. A. Sloman and D. A. Lagnado. Do we "do"? *Cognitive Science*, 29(1):5–39, Jan.-Feb. 2005. [43](#)
1462. P. Slovic. *The Perception of Risk*. Earthscan Publications Ltd, 2000. [142](#)
1463. P. E. Smaldino and R. McElreath. The natural selection of bad science. *Royal Society Open Science*, 3:160384, Aug. 2016. [9](#)
1464. G. K. Smith, A. A. Barbour, T. L. McNaugher, M. D. Rich, and W. L. Stanley. The use of prototypes in weapon system development. Report R-2345-AF, The RAND Corporation, Mar. 1981. [126](#)
1465. F. Söhnchen and S. Albers. Pipeline management for the acquisition of industrial projects. *Industrial Marketing Management*, 39(8):1356–1364, Nov. 2010. [119](#)
1466. M. Sojer, O. Alexy, S. Kleinknecht, and J. Henkel. Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code. *Journal of Management Information Systems*, 31(3):287–325, 2014. [120](#)
1467. Solganick & Co. Software M&A update. <http://www.solganickco.com/wp-content/uploads/2017/02/Solganick-Software-Q4-2016-final.pdf>, Apr. 2016. [83](#)
1468. G. S. Sommer. *Astronomical Odds A Policy Framework for the Cosmic Impact Hazard*. PhD thesis, Pardee RAND Graduate School, U.S.A., June 2004. [142](#)
1469. J. Sonnemanns. Price clustering and natural resistance points in the Dutch stock market: A natural experiment. *European Economic Review*, 50(8):1937–1950, Nov. 2006. [45](#)
1470. R. W. Soukoreff. *Quantifying Text Entry Performance*. PhD thesis, York University, Toronto, Canada, Apr. 2010. [19](#)
1471. SPEC. Standard performance evaluation corporation. <http://spec.org>, July 2014. [195](#), [197](#), [246](#), [286](#), [346](#)
1472. SPEC. SPEC power\_ssj 2008. [http://spec.org/power\\_ssj2008](http://spec.org/power_ssj2008), June 2016. [293](#)
1473. SPECpower Committee. Power and performance benchmark methodology. V 2.2, Standard Performance Evaluation Corporation (SPEC), Dec. 2014. [349](#)
1474. I. Spence and S. Lewandowsky. Displaying proportions and percentages. *Applied Cognitive Psychology*, 5(1):61–77, Apr. 1991. [202](#), [203](#)
1475. M. Spence. Job market signalling. *The Quarterly Journal of Economics*, 87(3):355–374, Aug. 1973. [66](#)
1476. D. Sperber and D. Wilson. *Relevance: Communication and Cognition*. Blackwell Publishers, second edition, 1995. [39](#), [163](#)
1477. D. Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, 2003. [166](#)
1478. D. Spinellis, V. Karakoidas, and P. Louridas. Comparative language fuzz testing: Programming languages vs. fat fingers. In *Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2012, pages 25–34, Oct. 2012. [152](#), [154](#)
1479. J. Spolsky. Fog Creek professional ladder. <https://www.joelonsoftware.com/2009/02/13/fog-creek-professional-ladder>, Feb. 2009. [64](#)
1480. J. Sprouse and D. Almeida. Assessing the reliability of textbook data in syntax: Adger's core syntax. *Journal of Linguistics*, 48(3):609–652, Nov. 2012. [151](#)
1481. D. Spuler. *C++ and C debugging, testing and reliability*. Prentice-Hall, Inc, 1994. [166](#)
1482. L. R. Squire and A. J. O. Dede. Conscious and unconscious memory systems. *Perspectives in Biology*, 7(3):a021667, Mar. 2015. [26](#)
1483. J. Srinivasan. *Lifetime Reliability Aware Microprocessor*. PhD thesis, University of Illinois at Urbana-Champaign, Oct. 2006. [155](#)
1484. E. B. Staats. Millions in savings possible in converting programs from one computer to another. Technical Report FGMSD-77-34, Office of Management and Budget, National Bureau of Standards, Sept. 1977. [102](#)
1485. C. B. Stabell and Ø. D. Fjeldstad. Configuring value for competitive advantage: On chains, shops, and networks. *Strategic Management Journal*, 19(5):413–437, May 1998. [76](#)
1486. Standish Group. The CHAOS report. Technical report, The Standish Group International, Inc, Aug. 1994. [112](#)
1487. P. Stanley-Marbell, V. Estellers, and M. Rinard. Crayon: Saving power through shape and color approximation on next-generation displays. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, page 11, Apr. 2016. [348](#)
1488. K. E. Stanovich. *Who Is Rational? Studies of Individual Differences in Reasoning*. Lawrence Erlbaum Associates, 1999. [39](#), [40](#)
1489. K. E. Stanovich and R. F. West. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and Brain Sciences*, 23(5):645–726, Oct. 2000. [39](#)
1490. M. Staples, R. Kolanski, G. Klein, C. Lewis, J. Andronick, T. Murray, R. Jeffery, and L. Bass. Formal specifications better than function points for code sizing. In *International Conference on Software Engineering*, ICSE'13, pages 1257–1260, May 2013. [194](#)
1491. J. Starek. A large-scale analysis of Java API usage. Thesis (m.s.), Institut für Informatik, Universität Koblenz-Landau, Mar. 2010. [280](#)
1492. E. Starr. The use, abuse, and enforceability of non-compete and no-poach agreements: A brief review of the theory, evidence, and recent reform efforts. Issue brief, Economic Innovation Group, Washington D.C., Feb. 2019. [98](#)
1493. T. N. Starr, L. K. Picton, and J. W. Thornton. Alternative evolutionary histories in the sequence space of an ancient protein. *Nature*, 549:409–413, Sept. 2017. [86](#)
1494. M. Stasinopoulos, B. Rigby, V. Voudouris, G. Heller, and F. D. Bastiani. Flexible regression and smoothing: The GAMLSS packages in R. draft book, July 2015. [280](#)
1495. G. Stasser and W. Titus. Effects of information load and percentage of shared information on the dissemination of unshared information during group discussion. *Journal of Personality and Social Psychology*, 53(1):81–93, July 1987. [72](#)
1496. M. Steele and J. Chaseling. Powers of discrete goodness-of-fit test statistics for a uniform null against a selection of alternative distributions. *Communications in Statistics-Simulation and Computation*, 35(4):1067–1075, Apr. 2006. [221](#)
1497. R. G. Steen, A. Casadevall, and F. C. Fang. Why has the number of scientific retractions increased? *PLoS ONE*, 8(7), Apr. 2013. [9](#)
1498. J. Steffens. *Newgames: Strategic Competition in the PC revolution*. Pergamon Press, 1994. [84](#)
1499. T. Stengos and E. Zacharias. Intertemporal pricing and price discrimination: A semiparametric hedonic analysis of the personal computer market. Discussion Paper 2002-11, Department of Economics, University of Cyprus, June 2002. [78](#)
1500. K. Stenning and M. van Lambalgen. Semantics as a foundation for psychology: A case study of Wason's selection task. *Journal of Logic, Language and Information*, 10(3):273–317, June 2001. [39](#)
1501. K. Stenning and M. van Lambalgen. A little logic goes a long way: basing experiment on semantic theory in the cognitive science of conditional reasoning. *Cognitive Science*, 28(4):481–530, July-Aug. 2004. [40](#)

1502. K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, 2008. 39
1503. M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, Sept. 1974. 218
1504. R. J. Sternberg and E. M. Weil. An aptitude-strategy interaction in linear syllogistic reasoning. Technical Report 15, Department of Psychology, Yale University, Apr. 1979. 40
1505. S. Sternberg. Memory-scanning: Mental processes revealed by reaction-time experiments. *American Scientist*, 57(4):421–457, 1969. 28
1506. A. Stevens and P. Coupe. Distortions in judged spatial relations. *Cognitive Psychology*, 10(4):422–437, Oct. 1978. 37
1507. N. Stewart, N. Chater, and G. D. A. Brown. Decision by sampling. *Cognitive Psychology*, 53(1):1–26, Jan. 2006. 49
1508. N. Stewart, C. Ungemach, A. J. L. Harris, D. M. Bartels, B. R. Newell, G. Paolacci, and J. Chandler. The average laboratory samples a population of 7,300 Amazon Mechanical Turk workers. *Judgment and Decision Making*, 10(5):479–491, Sept. 2015. 341
1509. G. Stikkel. Dynamic model for the system testing process. *Information and Software Technology*, 48(7):578–585, July 2006. 160
1510. V. Stodden, P. Guo, and Z. Ma. Toward reproducible computational research: An empirical analysis of data and code policy adoption by journals. *PLoS ONE*, 8(6):e13636, June 2013. 8
1511. G. P. Stone, D. B. Levin, H. Hwang, M. Kim, and C. Mckay. JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, case no. 1-05-CV-039231, filing #g-64475. Opinion, Superior court of the state of California for the county of Santa Clara, 2014. 346
1512. J. Stone, M. Greenwald, C. Partridge, and J. Hughes. Performance of checksums and CRCs over real data. *IEEE/ACM Transactions on Networking*, 6(5):529–543, Oct. 1998. 140
1513. D. Straker. *C-Style standards and guidelines*. Prentice-Hall, Inc, 1992. 166
1514. S. Strand, I. J. Deary, and P. Smith. Sex differences in cognitive abilities test scores: A UK national picture. *British Journal of Educational Psychology*, 76(3):463–480, Apr. 2006. 17
1515. W. Stroebe, B. A. Nijstad, and E. F. Rietzschel. Beyond productivity loss in brainstorming groups: The evolution of a question. Working Paper No. 2014-05, Center for Research in Economics, Management and the Arts, CREMA Südstrasse 11 CH, 2014. 72
1516. H. Strong. Mozilla foundation and subsidiary december 31, 2015 and 2014. Independent auditors' report and consolidated financial statements, Hood & Strong LLC, Nov. 2016. 111
1517. R. Sudan, S. Ayers, P. Dongier, A. Muente-Kunigami, and C. Z.-W. Qiang. The global opportunity in IT-based services: Assessing and enhancing country competitiveness. Report, The World Bank, 2010. 56
1518. C. Sun, V. Le, Q. Zhang, and Z. Su. Toward understanding compiler bugs in GCC and LLVM. In *Proceedings of the 2016 International Symposium on Software Testing and Analysis*, ISSTA'16, pages 294–305, July 2016. 149, 151
1519. L. Sun. What we are paying for: A quality adjusted price index for laptop microprocessors. Honors thesis, Wellesley College, Apr. 2014. 76, 77
1520. Sun Microsystems, Inc. Java code conventions. Technical report, Sun Microsystems, Inc, Sept. 1997. 186, 335
1521. T. Sunada, A. Monden, and K. Matsumoto. On estimating source lines of code from a binary program. In *Joint Conference of International Workshop on Software Measurement and International Conference on Software Process and Product Measurement*, IWSM/Mensura 2011, pages 3–6, Nov. 2011. 165
1522. C. R. Sunstein. *The Cost-Benefit Revolution*. The MIT Press, Aug. 2018. 137
1523. A. Suresh, B. N. Swamy, E. Rohou, and A. Seznec. Intercepting functions for memoization: A case study using transcendental functions. *Transactions on Architecture and Code Optimization*, 12(2):18, July 2015. 182
1524. K. Suzuki and S. Swanson. A survey of trends in non-volatile memory technologies: 2000–2014. In *IEEE International Memory Workshop*, IMW, pages 1–4, May 2015. 349
1525. T. N. Suzuki, D. Wheatcroft, and M. Griesser. Experimental evidence for compositional syntax in bird calls. *Nature Communications*, 7(10986), Mar. 2016. 16
1526. M. Swan and B. Smith. *Learner English: A teacher's guide to interference and other problems*. Cambridge University Press, second edition, 2001. 175
1527. G. M. Swift and S. M. Guertin. In-flight observations of multiple-bit upset in DRAMs. *IEEE Transactions on Nuclear Science*, 47(6):2386–2391, Dec. 2000. 156
1528. R. A. Syed, B. Robinson, and L. Williams. Does hardware configuration and processor load impact software fault observability? In *Third International Conference on Software Testing, Verification and Validation*, ICST, pages 285–294, Apr. 2010. 236, 237
1529. I. H. Tabernero. *A statistical examination of the properties and evolution of libre software*. PhD thesis, Universidad Rey Juan Carlos, Oct. 2008. 166, 261, 312
1530. N. Taerat, N. Nakshinehaboon, C. Chandler, J. Elliott, C. B. Leangsuksun, G. Ostrouchov, S. L. Scott, and C. Englemann. Blue Gene/L log analysis and time to interrupt estimation. In *International Conference on Availability, Reliability and Security*, ARES'09, pages 173–180, Oct. 2009. 365
1531. L. Takeyama. The shareware industry: Some stylized facts and estimates of rates of return. *Economics of Innovation and New Technology*, 3(2):161–174, Jan. 1994. 76
1532. P. P. Tallon, R. J. Kauffman, H. C. Lucas, A. B. Whinston, and K. Zhu. Using real options analysis for evaluating uncertain investments in information technology: Insights from the ICIS 2001 debate. *Communications of the Association for Information Systems*, 9:136–167, Sept. 2002. 109
1533. K. Y. Tam. Capital budgeting in information systems development. *Information & Management*, 23(6):345–357, Dec. 1992. 56
1534. T. Tamai and Y. Torimitsu. Software lifetime and its evolution process over generations. In *Proceedings of 1992 Conference on Software Maintenance*, pages 63–69, Nov. 1992. 59, 89, 90
1535. P.-N. Tan and V. K. J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, June 2004. 256
1536. E. Tang, E. Barr, X. Li, and Z. Su. Perturbing numerical calculations for statistical analysis of floating-point program (in)stability. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA'10, pages 131–142, July 2010. 139
1537. V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX Annual Technical Conference*, ATC'12, June 2012. 226, 341
1538. R. C. Tausworthe. Staffing implications of software productivity models. In E. C. Posner, editor, *The Telecommunication and Data Acquisition Report 42-72*, pages 70–77. Jet Propulsion Laboratory, California Institute of Technology, Oct.–Dec. 1982. 133
1539. Q. C. Taylor. Analysis and characterization of author contribution patterns in open source software development. Thesis (m.s.), Brigham Young University, Apr. 2012. 179
1540. M. Tedre. Computing as a science: A survey of competing viewpoints. *Minds & Machines*, 21(3):361–387, Aug. 2011. 5
1541. J. Teixeira, G. Robles, and J. M. González-Barahona. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. *Journal of Internet Services and Applications*, 6(1):1–27, 2015. 73
1542. M. Templ, B. Meindl, and A. Kowarik. Introduction to statistical disclosure control (SDC). Technical report, International Household Survey Network, Oct. 2015. 358
1543. K. Tentori, D. Osherson, L. Hasher, and C. May. Wisdom and ageing: Irrational preferences in college students but not older adults. *Cognition*, 81(3):B87–B99, 2001. 48
1544. P. E. Tetlock. Accountability: The neglected social context of judgment and choice. *Research in Organizational Behavior*, 7:297–332, 1985. 49
1545. P. E. Tetlock. An alternative metaphor in the study of judgment and choice: People as politicians. *Theory and Psychology*, 1(4):451–475, 1991. 49

1546. Tezzaron Semiconductor. Soft errors in electronic memory. Technical Report 1.1, Tezzaron Semiconductor, Naperville, IL, Jan. 2004. 155
1547. T. A. Thayer, M. Lipow, and E. C. Nelson. *Software Reliability*. North-Holland Publishing Company, 1978. 5, 140
1548. E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel. Practical performance models for complex, popular applications. In *Performance Evaluation Review*, SIGMETRICS'10, pages 1–12, June 2010. 201, 202
1549. D. R. Thomas. *Security metrics for computer systems*. PhD thesis, Cambridge Computer Laboratory, University of Cambridge, Sept. 2015. 139, 143
1550. M. Thomas and V. Morwitz. Penny wise and pound foolish: The left-digit effect in price cognition. *Journal of Consumer Research*, 32(1):54–64, June 2005. 77
1551. M. Thomas, D. H. Simon, and V. Kadiyali. Do consumers perceive precise prices to be lower than round prices? Evidence from laboratory and market data. Research Paper Series #09-07, Johnson School, Cornell University, Sept. 2007. 77
1552. B. Thompson. The bill gates line. Blog, May 2018. <https://stratechery.com/2018/the-bill-gates-line/>. 98
1553. P. Thompson. How much did the Liberty shipbuilders forget? *Management Science*, 53(6):908–918, June 2007. 69
1554. S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, 15(1):1–34, Feb. 2010. 6, 118
1555. J. D. Tinder. Entry granting reasserted motion to dismiss (docket no. 34): Daniel wallace, v. free software foundation, inc. Case 1:05-cv-0618-JDT-TAB, UNITED STATES DISTRICT COURT SOUTHERN DISTRICT OF INDIANA INDIANAPOLIS DIVISION, Mar. 2006. 64
1556. M. A. Tinker. The relative legibility of the letters, the digits, and of certain mathematical signs. *Journal of Generative Psychology*, 1:472–494, 1928. 176
1557. J. T. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1A):40–50, 1971. 19
1558. T. S. Traen. The Brooks Act: An 8-bit act in a 64-bit world? An investigation of the Brooks Act and its implications to the department of defense information technology acquisition process. Executive Research Project S18, The Industrial College of the Armed Forces, National Defense University, Washington, D.C., May 1995. 94
1559. Transport, Department for. The accidents sub-objective. Transport Analysis Guidance Unit 3.4.1, Department for Transport, United Kingdom, Apr. 2011. 142
1560. A. Treisman and J. Souther. Search asymmetry: A diagnostic for preattentive processing of separable features. *Journal of Experimental Psychology: General*, 114(3):285–310, 1985. 23, 24
1561. L. M. Trick and Z. W. Pylyshyn. What enumeration studies can show us about spatial attention: Evidence for limited capacity preattentive processing. *Journal of Experimental Psychology: Human Perception and Performance*, 19(2):331–351, 1993. 43
1562. J. E. Triplett. Performance measures for computers. In *Deconstructing the Computer*, pages 99–139, Feb. 2003. 1
1563. K. S. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, Inc, second edition, 2002. 227
1564. J. S. Trueblood and J. R. Busemeyer. A quantum probability account of order effects in inference. *Cognitive Science*, 35(8):1518–1552, Nov.-Dec. 2011. 34
1565. C.-C. Tsai, B. Jain, N. A. Abdul, and D. E. Porter. A study of modern Linux API usage and compatibility: What to support when you're supporting. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, page 16, Apr. 2016. 105
1566. N. P. Tschacher. Typosquatting in programming language package managers. Thesis (b.sc.), Department of Informatics, University of Hamburg, Mar. 2016. 151
1567. T. K. Tsingos. Enforceability of free/open source software licensing terms: A critical review of the global case - law. In *Fourth International Conference on Information Law*, ICIL 2011, May 2011. 64
1568. TSMC. TSMC historical operating data. [http://www.tsmc.com/english/investorRelations/historical\\_information.htm](http://www.tsmc.com/english/investorRelations/historical_information.htm), May 2017. 85
1569. T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. In *Proceedings of Usability Professionals Association*, pages 1–12, June 2004. 354
1570. J. Turley. Embedded processors. <http://www.extremetech.com>, Jan. 2002. 84, 278
1571. H. Turner and D. Firth. Bradley-Terry models in R: The BradleyTerry2 package. *Journal of Statistical Software*, 48(9):1–21, 2012. 334
1572. H. Turner and D. Firth. *Generalized nonlinear models in R: An overview of the gnm package*. University of Warwick, UK, 1.0-8 edition, Apr. 2015. 297
1573. M. L. Turner and R. W. Engle. Is working memory capacity task dependent? *Journal of Memory and Language*, 28(2):127–154, Apr. 1989. 172
1574. R. Turner. *Weathering Heights: The Emergence of Aeronautical Meteorology as an Infrastructural Science*. PhD thesis, University of Pennsylvania, May 2010. 1, 82
1575. J. Tzelgov, V. Yehene, L. Kotler, and A. Alon. Automatic comparisons of artificial digits never compared: Learning linear ordering relations. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 26(1):103–120, 2000. 46
1576. UBM. Then, now: What's next? Embedded Market Study 2014, UBM Electronics, 2014. 103
1577. A. Čaušević, R. Shukla, S. Punnekkat, and D. Sundmark. Effects of negative testing on TDD: An industrial experiment. In H. Baumeister and B. Weber, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 149 of *Lecture Notes in Business Information Processing*, pages 91–105. Springer Berlin Heidelberg, 2013. 161
1578. The ultimate Debian database. website, 2014. <http://wiki.debian.org/Ultimate Debian Database/>. 140, 199, 275, 276
1579. Unicode Consortium, The. *The Unicode Standard Version 11.0 – Core Specification*. The Unicode Consortium, June 2018. 96
1580. D. Šmite, R. Britto, and R. van Solingen. Calculating the extra costs and the bottom-line hourly cost of offshoring. In *IEEE 12th International Conference on Global Software Engineering*, ICGSE'17, pages 96–105, May 2017. 116
1581. I. Utting, D. Bouvier, M. Caspersen, A. E. Tew, R. Frye, Y. B.-D. Kolikant, M. McCracken, J. Paterson, J. Sorva, L. Thomas, and T. Wilusz. A fresh look at novice programmers' performance and their teachers' expectations. In *ITiCSE-WGR'13*, pages 15–32, June 2013. 340
1582. Ž Antolić. Fault slip through measurement process implementation in CPP software verification. In *International Conference on Business Intelligence Systems*, miproBIS 2007, page ???, May 2007. 158
1583. A. Vahabzadeh, A. M. Fard, and A. Mesbah. An empirical study of bugs in test code. In *International Conference on Software Maintenance and Evolution*, ICSME 2015, pages 101–110, Oct. 2015. 160
1584. M. Välimäki. Dual licensing in open source software industry. *Systèmes d'Information et Management*, 8(1):63–75, 2003. 62
1585. J. van Angeren, C. Alves, and S. Jansen. Can we ask you to collaborate? Analyzing app developer relationships in commercial platform ecosystems. *Journal of Systems and Software*, 113(???:430–445, Mar. 2016. 81, 82
1586. O. Van den Bergh, S. Vrana, and P. Eelen. Letters from the heart: Affective categorization of letter combinations in typists and nontypists. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 16(6):1153–1161, 1990. 95
1587. K. G. van den Boogaart and R. Tolosana-Delgado. *Analyzing Compositional Data with R*. Springer, 2013. 327
1588. J.-B. van der Henst, L. Carles, and D. Sperber. Truthfulness and relevance in telling the time. *Mind & Language*, 17(5):457–466, Nov. 2002. 45
1589. E. van der Kouwe, D. Andriesse, H. Bos, C. Giuffrida, and G. Heiser. Benchmarking crimes: An emerging threat in systems security. In *eprint arXiv:cs.CR/1801.02381*, Jan. 2018. 346
1590. C. van der Merwe. An engineering approach to an integrated value proposition design framework. Thesis (m.s.), Faculty of Industrial Engineering at Stellenbosch University, Mar. 2015. 78
1591. M. J. P. van der Meulen. *The Effectiveness of Software Diversity*. PhD thesis, Centre for Software Reliability, City University, Nov. 2007. 118, 164, 167, 221, 222

1592. M. J. P. van der Meulen, P. G. Bishop, and M. Revilla. An exploration of software faults and failure behaviour in a large population of programs. In *15th International Symposium on Software Reliability Engineering*, ISSRE 2004, pages 101–120, Nov. 2004. [151](#), [153](#)
1593. M. P. van Oeffelen and P. G. Vos. A probabilistic model for the discrimination of visual number. *Perception & Psychophysics*, 32(2):163–170, 1982. [43](#)
1594. K. E. van Oorschot, J. W. M. Bertrand, and C. G. Rutte. Field studies into the dynamics of product development tasks. *International Journal of Operations & Production Management*, 25(8):720–739, 2005. [125](#)
1595. H. VanLehn. *Mind Bugs: The Origins of Procedural Misconceptions*. The MIT Press, 1990. [19](#), [44](#)
1596. R. Vasa. *Growth and Change Dynamics in Open Source Software Systems*. PhD thesis, Faculty of Information and Communication Technology, Swinburne University of Technology, Melbourne, Oct. 2010. [100](#), [236](#), [270](#)
1597. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload-A case study of the Gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, Aug. 2012. [276](#), [277](#)
1598. P. Vassiliadis, M.-R. Kolozoff, M. Zerva, and A. V. Zarras. Schema evolution and foreign keys: Birth, eviction, change and absence. In *International Conference on Conceptual Modeling*, ER 2017, pages 106–119, Nov. 2017. [188](#)
1599. VCDB. VERIS community database. <https://github.com/vz-risk/VCDB>, Mar. 2018. [141](#)
1600. W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, fourth edition, 2002. [311](#)
1601. C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. License usage and changes: A large-scale study on GitHub. *Empirical Software Engineering*, 22(3):1537–1577, June 2017. [62](#), [63](#)
1602. P. Verghese and D. G. Pelli. The information capacity of visual attention. *Vision Research*, 32(5):983–995, 1992. [52](#)
1603. C. Verhoef. Quantitative IT portfolio management. *Science of Computer Programming*, 45(1):1–96, Oct. 2002. [115](#)
1604. I. Vessey. Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, 22(2):219–240, Mar. 1991. [202](#)
1605. A. Vetrò, N. Zazworska, C. Seaman, and F. Shull. Using the ISO/IEC 9126 product quality model to classify defects: a controlled experiment. In *16th International Conference on Evaluation & Assessment in Software Engineering*, EASE 2012, pages 187–196, May 2012. [141](#), [304](#)
1606. B. Veysman and L. Akhmadeeva. Towards evidence-based typography: First results. *TUGboat*, 33(2):156–156, Apr. 2012. [218](#), [219](#)
1607. Vgchartz global yearly chart: 2005–2016. website, Feb. 2017. <http://www.vgchartz.com/yearly/2016/Global/>. [78](#)
1608. V. B. Viard. Information goods upgrades: Theory and evidence. *The B.E. Journal of Theoretical Economics*, 7(1):1–34, 2007. [77](#), [78](#)
1609. C. Vickrey and A. Neuringer. Pigeon reaction time, Hick’s law, and intelligence. *Psychonomic Bulletin & Review*, 7(2):284–291, June 2000. [53](#)
1610. N. M. Victor and J. H. Ausubel. DRAMs as model organisms for study of technological evolution. *Technological Forecasting & Social Change*, 69(3):243–262, Apr. 2002. [82](#)
1611. S. Vidal, A. Bergel, J. A. Díaz-Pace, and C. Marcosa. Over-exposed classes in Java: An empirical study. *Computer Languages, Systems & Structures*, 46:1–19, Nov. 2016. [186](#)
1612. F. Viénot, H. Brettel, and J. D. Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *COLOR research and application*, 24(4):243–252, Aug. 1999. [206](#)
1613. V. Villard. Android version distribution history. <http://www.bidouille.org/misc/androidcharts>, 2015. [88](#), [90](#), [207](#)
1614. T. H. Vines, A. Y. K. Albert, R. L. Andrew, F. Débarre, D. G. Bock, M. T. Franklin, K. J. Gilbert, J.-S. Moore, S. Renaut, and D. J. Renison. The availability of research data declines rapidly with article age. In *eprint arXiv:abs/1312.5670*, Dec. 2013. [8](#)
1615. W. K. Viscusi and J. E. Aldy. The value of a statistical life: A critical review of market estimates throughout the world. Working Paper No. 9487, National Bureau of Economic Research, USA, Feb. 2003. [141](#)
1616. R. Viseur and G. Robles. First results about motivation and impact of license changes in open source projects. In *11th IFIP WG 2.13 International Conference*, OSS 2015, pages 137–145, May 2015. [63](#)
1617. J. Volstorf. *Against all noise: On noise-robust strategies in the emergence of cooperation*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät II, Humboldt-Universität, Feb. 2013. [71](#)
1618. K. G. Volz and G. Gigerenzer. Cognitive processes in decisions under risk are not the same as in decisions under uncertainty. *frontiers in Neuroscience*, 6:105, July 2012. [48](#)
1619. K. von Fintel and L. Mattheusson. Universals in semantics. *The Linguistic Review*, 25(1-2):139–201, 2008. [38](#)
1620. A. von Rhein, J. Liebig, A. Janker, C. Kästner, and S. Apel. Variability-aware static analysis at scale: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 27(4):???, Nov. 2018. [154](#)
1621. S. L. R. Vrhovec, T. Hovelja, D. Vavpotič, and M. Krisper. Diagnosing organizational risks in software projects: Stakeholder resistance. *International Journal of Project Management*, 33(6):1262–1273, Aug. 2015. [127](#)
1622. M. Wachs. When planners lie with numbers. *Journal of the American Planning Association*, 55(4):476–479, Apr. 1989. [113](#)
1623. J. Wagemans, J. H. Elder, M. Kubovy, M. A. Peterson, S. E. Palmer, M. Singh, and R. von der Heydt. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure-ground organization. *Psychological Bulletin*, 138(6):1172–1217, 2012. [23](#)
1624. J. Wai, M. Cacchio, M. Putallaz, and M. C. Makel. Sex differences in the right tail of cognitive abilities: A 30 year examination. *Intelligence*, 38(4):412–423, July-Aug. 2010. [17](#)
1625. J. Wainer, C. G. N. Barsottini, D. Lacerda, and L. R. M. de Marco. Empirical evaluation in computer science research published by ACM. *Information and Software Technology*, 51(6):1081–1085, June 2009. [6](#)
1626. L. Wakeham. Government policy on the management of risk, volume I: Report. HL Paper 183-I, Select Committee on Economic Affairs, UK House of Lords, June 2006. [142](#)
1627. S. Waligora, J. Bailey, and M. Stark. Impact of Ada and object-oriented design in the flight dynamics division at Goddard space flight center. Technical Report SEL-95-001, Goddard Space Flight Center, Mar. 1995. [167](#), [194](#)
1628. D. R. Wallace and D. R. Kuhn. Failure modes in medical device software: An analysis of 15 years of recall data. *International Journal of Reliability, Quality and Safety Engineering*, 8(4):351–372, Dec. 2001. [141](#)
1629. H. Wang, H. Li, L. Li, Y. Guo, and G. Xu. Why are Android apps removed from Google play? A large-scale empirical study. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR’18, pages 231–242, May 2018. [99](#)
1630. P. Wang. Chasing the hottest IT: Effects of information technology fashion on organizations. *MIS Quarterly*, 34(1):63–85, Mar. 2010. [4](#), [8](#)
1631. W. Wang. Toward improved understanding and management of software clones. Thesis (m.s.), University of Waterloo, Ontario, Canada, May 2012. [131](#), [189](#)
1632. Y. Wang. Language matters. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM’15, page ???, Oct. 2015. [96](#)
1633. Y. Wang and J. Zhang. The effort distribution of software development phases. *Computer Science and Application*, 7(5):428–437, May 2017. [114](#)
1634. L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. A case for opportunistic embedded sensing in presence of hardware power variability. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower’10, pages 1–8, Oct. 2010. [349](#)
1635. G. Ward, L. Tan, and R. Grenfell-Essam. Examining the relationship between free recall and immediate serial recall: the effects of list length and output order. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 36(5):1207–1241, Sept. 2010. [30](#)
1636. P. J. Ward. Euclid’s Elements, from Hilbert’s axioms. Thesis (m.s.), The Ohio State University, 2012. [138](#)

1637. C. Ware. *Information Visualization Perception for Design*. Morgan Kaufmann Publishers, 2000. 22
1638. W. H. Ware, S. N. Alexander, P. Armer, M. M. Astrahan, L. Bers, H. H. Goode, H. D. Huskey, and M. Rubinoff. Soviet computer technology—1959. Research Memorandum RM-2541, The RAND Corporation, Mar. 1960. 5
1639. P. C. Wason. On the failure to eliminate hypotheses in a conceptual task. *The Quarterly Journal of Experimental Psychology*, XII(3):129–140, 1960. 21
1640. P. C. Wason. Reasoning about a rule. *The Quarterly Journal of Experimental Psychology*, 20(3):273–281, 1968. 39, 40
1641. J. Waters. Variable marginal propensities to pirate and the diffusion of computer software. MPRA Paper No. 46036, Nottingham University Business School, Apr. 2013. 78
1642. C. Watson and F. W. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation technology in computer science education*, ITICSE’14, pages 39–44, June 2014. 340
1643. V. M. Weaver and J. Dongarra. Can hardware performance counters produce expected, deterministic results? In *3rd Workshop on Functionality of Hardware Performance Monitoring*, pages 1–11, Dec. 2010. 349
1644. V. M. Weaver and S. A. McKee. Can hardware performance counters be trusted? In *IEEE International Symposium on Workload Characterization*, IISWC’08, pages 141–150, Sept. 2008. 349
1645. E. U. Weber, A.-R. Blais, and N. E. Betz. A domain-specific risk-attitude scale: Measuring risk perceptions and risk behaviors. *Journal of Behavior and Decision Making*, 15(4):263–290, Apr. 2002. 48
1646. B. F. Webster. Patterns in IT litigation: Systems failure (1976-2000). A study, PriceaterhouseCoopers LLP, 2000. 120
1647. B. S. Weekes. Differential effects of number of letters on word and nonword naming latency. *The Quarterly Journal of Experimental Psychology*, 50A(2):439–456, 1997. 28
1648. D. M. Wegner. *The Illusion of Conscious Will*. MIT Press, 2002. 16
1649. M. H. Weik. A survey of domestic electronic digital computing systems. Technical Report 971, Ballistic Research Laboratories, Maryland, Dec. 1955. 102, 345
1650. M. H. Weik. A third survey of domestic electronic digital computing systems. Technical Report 1115, Ballistic Research Laboratories, Maryland, Mar. 1961. 102, 345
1651. G. F. Weinwurm and H. J. Zagorski. Research into the management of computer programming: A transitional analysis of cost estimation techniques. Technical Documentary Report ESD-TR-65-575, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Nov. 1965. 115
1652. M. V. Welser. Opposing the monetization of Linux: McHardy v. Geinatech & addressing copyright "trolling" in Germany. *International Free and Open Source Software Law Review*, 10(1):9–20, 2018. 63
1653. J. West and J. Dedrick. Innovation and control in standards architectures: The rise and fall of Japan’s PC-98. *Information Systems Research*, 11(2):197–216, June 2000. 86
1654. J. A. White. Grapher pics. <http://www.talljerome.com/mathnerd.html>, Oct. 2012. 209
1655. M. White. Scaled CMOS technology reliability users guide. JPL Publication 09-33 01/10, Jet Propulsion Laboratory, California Institute of Technology, 2010. 155
1656. White House, The. Guidelines and discount rates for benefit-cost analysis of federal programs. OMB Circular A-94, U.S. Government, 1992. 58
1657. D. Whitfield. Cost overruns, delays and terminations: 105 outsourced public sector ICT projects. ESSU Research Report 3, European Services Strategy Unit, Dec. 2007. 112, 113
1658. R. M. Whyte. Order Re Sun’s Motions for Preliminary Injunction Against Microsoft. Re: Sun Microsystems v. Microsoft, Case No. 97-20884 RMW(PVT). Opinion, UNITED STATES DISTRICT COURT FOR THE NORTHERN DISTRICT OF CALIFORNIA, 1998. 100
1659. J. M. Wicherts, M. Bakker, and D. Molenaar. Willingness to share research data is related to the strength of the evidence and the quality of reporting of statistical results. *PLoS ONE*, 6(11):e26828, Nov. 2011. 8
1660. W. A. Wickelgren. Size of rehearsal group and short-term memory. *Journal of Experimental Psychology*, 68(4):413–419, 1964. 30
1661. G. Wiederhold. What is your software worth? Technical Report ???, Stanford University, Apr. 2007. 75
1662. A. Wierzbicka. *Semantics: Primes and Universals*. Oxford University Press, 1996. 38
1663. Wikipedia. List of most expensive video games to develop. website, 2018. [https://en.wikipedia.org/List\\_of\\_most\\_expensive\\_video\\_games\\_to\\_develop](https://en.wikipedia.org>List_of_most_expensive_video_games_to_develop). 57
1664. R. Wilcox. *Introduction to Robust Estimation & Hypothesis Testing*. Elsevier, 3rd edition, 2012. 242
1665. J. Wiley. Expertise as mental set: The effects of domain knowledge in creative problem solving. *Memory & Cognition*, 26(4):716–730, 1998. 35
1666. M. V. Wilkes. *Memoirs of a Computer Pioneer*. MIT Press, 1984. 137
1667. M. V. Wilkes, D. J. Wheeler, and S. Gill. *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley, second edition, 1957. 104
1668. L. Wilkinson. *The Grammar of Graphics*. Springer, second edition, 2005. 203
1669. J. C. Williams. A data-based method for assessing and reducing human error to improve operational performance. In *Fourth Conference on Human Factors and Power Plants*, pages 436–450, June 1988. 19
1670. P. Williams and B. Curtis. A matched project evaluation of modern programming practices: Scientific report on the ASTROS plan. Technical Report RADC-TR-80-6, Vol II, General Electric Company, Feb. 1980. 132
1671. R. R. Willis, R. M. Rova, M. D. Scott, M. I. Johnson, J. F. Ryskowski, J. A. Moon, K. C. Shumate, and T. O. Winfield. Hughes Aircraft’s widespread deployment of a continuously improving software process. Technical Report CMU/SEI-98-TR-006, Raytheon Systems Company, May 1998. 74
1672. H. E. Willman, Jr., T. A. James, A. A. Beauregard, and P. Hilcoff. Software systems reliability: A Raytheon project history. Final Technical Report RADC-TR-77-188, Rome Air Development Center, Griffiss Air Force Base, June 1977. 140
1673. L. M. Wills. Automated program recognition by graph parsing. A.I. Technical Report No. 1358, MIT Artificial Intelligence Laboratory, July 1992. 182
1674. C. Wohlin, P. Runeson, and J. Brantestam. An experimental evaluation of capture-recapture in software inspections. *Journal of Software Testing, Verification and Reliability*, 5(4):213–232, 1995. 304
1675. R. W. Wolverton. The cost of developing large-scale software. *IEEE Transactions on Computers*, c-23(6):615–636, June 1974. 123
1676. W. E. Wong, S. S. Gokhale, and J. R. Horgan. Quantifying the closeness between program components and features. *The Journal of Systems and Software*, 54(2):87–98, Oct. 2000. 152, 182, 183
1677. A. Wood. Software reliability growth models. Technical Report 96.1, Tandem Computer, Sept. 1996. 147, 148
1678. R. Woodfield. Undergraduate retention and attainment across the disciplines. Report, The Higher Education Academy, York, UK, Dec. 2014. 340
1679. Semiconductor monthly sales volume: 1975-2016. website, Mar. 2016. <https://www.wsts.org>. 4
1680. D. Wren. Passmark website. <http://www.passmark.com>, July 2014. 354, 355
1681. R. Wright. *The Evolution of GOD*. Little, Brown Book Group, 2009. 86
1682. S. D. Wu, C. Rossin, K. G. Kempf, M. O. Atan, B. Aytac, S. A. Shirodkar, and A. Mishra. Extending Bass for improved new product forecasting. ???, ???(??):???, Apr. 2009. 78
1683. J. Yan and W. Zhang. Compiler-guided register reliability improvement against soft errors. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT’05, pages 203–209, Sept. 2005. 156
1684. M. Yang, G.-R. Uh, and D. B. Whalley. Efficient and effective branch reordering using profile data. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(6):667–697, Nov. 2002. 184
1685. M. C. K. Yang and A. Chao. Reliability-estimation & stopping-rules for software testing, based on repeated appearances of bugs. *IEEE Transactions on Reliability*, 44(2):315–321, June 1995. 162

1686. X. Yang, Z. Wang, J. Xue, and Y. Zhou. The reliability wall for exascale supercomputing. *IEEE Transactions on Computers*, 61(6):767–779, June 2011. [157](#)
1687. M. J. Yap, S. J. R. Liow, S. B. Jalil, and S. S. B. Faizal. The Malay lexicon project: A database of lexical statistics for 9,592 words. *Behavior Research Methods*, 42(4):992–1003, Nov. 2010. [175](#)
1688. Y. C. B. Yeh. Triple-triple redundant 777 primary flight computer. In *Proceedings Aerospace Applications Conference (vol 1)*, pages 293–307, Feb. 1996. [156](#)
1689. J. R. Yost. *Making IT Work: A History of the Computer Services Industry*. The MIT Press, 2017. [94](#)
1690. A. G. Yu. *Managing Application Software Suppliers in Information System Development Projects*. PhD thesis, Department of Management and Organisation, University of Stirling, Nov. 2003. [121](#)
1691. L. Yu and S. Ramaswamy. A study of SourceForge users and user network. AAAI Technical Report FS-13-05, Association for the Advancement of Artificial Intelligence, Nov. 2013. [182](#)
1692. D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE’12, pages 102–112, June 2012. [157](#)
1693. T. Yuki and S. Rajopadhye. Folklore confirmed: Compiling for speed = compiling for energy. Technical Report CS13-107, Computer Science Department, Colorado State University, Aug. 2013. [347](#)
1694. A. Zaidman, B. V. Rompaey, A. van Deursen, and S. Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. Technical Report TUD-SERG-2010-035, Software Engineering Research Group, Delft University of Technology, 2010. [160](#)
1695. S. F. Zeigler. Comparing development costs of C and Ada. Technical report, Rational Software Corporation, Mar. 1995. [52](#)
1696. A. Zeileis, K. Hornik, and P. Murrell. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, July 2009. [206](#)
1697. M. V. Zelkowitz. The effectiveness of software prototyping: A case study. In *ACM Washington Chapter 26th Annual Technical Symposium*, pages 7–15, June 1987. [126](#)
1698. A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word—A parody in empirical research-. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, PROMISE’11, pages 5:1–5:7, Sept. 2011. [260](#)
1699. J. Zhang and H. Wang. The effect of external representations on numeric tasks. *The Quarterly Journal of Experimental Psychology*, 58(5):817–838, Oct. 2005. [44](#)
1700. J. Zhang, M. Zhu, D. Hao, and L. Zhang. An empirical study on the scalability of selective mutation testing. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 277–287, Nov. 2014. [162](#)
1701. X. Zhang. *An Analysis of the Effect of Environmental and Systems Complexity on Information Systems Failures*. PhD thesis, University of North Texas, Aug. 2001. [92](#)
1702. Y. Zhang, Y. Jiang, C. Xu, X. Ma, and P. Yu. ABC: Accelerated building of C/C++ projects. In *Asia-Pacific Software Engineering Conference*, APSEC 2015, pages 182–189, Dec. 2015. [187](#)
1703. Y. Zhang, J. W. Lee, N. P. Johnson, and D. I. August. DAFT: Decoupled acyclic fault tolerance. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT’10, pages 87–98, Sept. 2010. [156](#)
1704. M. Zhao, J. Grossklags, and P. Liu. An empirical study of web vulnerability discovery ecosystems. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS’15, pages 1105–1117, Oct. 2015. [98](#), [141](#)
1705. M. Zhao and P. Liu. Empirical analysis and modeling of black-box mutational fuzzing. In *International Symposium on Engineering Secure Software and Systems*, ESSoS 2016, pages 173–189, Apr. 2016. [147](#), [149](#)
1706. Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: A large-scale empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE’17, pages 60–71, Oct.-Nov. 2017. [130](#)
1707. J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk. On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, 32(4):240–253, Apr. 2006. [159](#)
1708. Q. Zheng, A. Mockus, and M. Zhou. A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE’15, pages 637–648, Aug.-Sept. 2015. [360](#)
1709. H. Zhong and Z. Su. An empirical study on real bug fixes. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE’15, pages 913–923, May 2015. [154](#)
1710. H. Zhou and A. Fishbach. The pitfall of experimenting on the web: How unattended selective attrition leads to surprising (yet false) research conclusions. *Journal of Personality and Social Psychology*, 111(4):493–504, Oct. 2016. [341](#)
1711. K. Zhou, P. Huang, C. Li, and H. Wang. An empirical study on the interplay between filesystems and SSD. In *7th International Conference on Networking, Architecture and Storage*, NAS, pages 124–133, June 2012. [352](#), [353](#)
1712. X. Zhu, E. J. Whitehead, Jr., C. Sadowski, and Q. Song. An analysis of programming language statement frequency in C, C++, and Java source code. *Software—Practice and Experience*, 15(11):1479–1495, Nov. 2015. [220](#)
1713. A. Ziegler, V. Rothberg, and D. Lohmann. Analyzing the impact of feature changes in Linux. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS’16, pages 25–32, Jan. 2016. [183](#)
1714. T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE 2009, pages 91–100, Aug. 2009. [158](#)
1715. T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE’07, May 2007. [153](#)
1716. P. M. Zislis. An experiment in algorithm implementation. Technical Report CSD-TR 96, Purdue University, June 1973. [33](#), [34](#), [167](#), [178](#)
1717. F. Zlotnick. *The POSIX.1 Standard: A Programmer’s Guide*. The Benjamin/Cummings Publishing Company, 1991. [104](#)
1718. K. Zuse. Über den allgemeinen Plankalkül als mittel zur formulierung schematisch-kombinativer aufgaben. *Archiv der Mathematik*, 1:441–449, 1949. [102](#)
1719. O. Zwikaal and S. Globerson. Benchmarking of project planning and success in selected industries. *Benchmarking: An International Journal*, 13(6):688–700, 2006. [111](#)