Introduction to Software Engineering

Chapter One - Part One

CMPT 276

Sommerville, 10th Ed. supplemented by Steven Pearce and the 2014 ACM Guidelines for SWE

Objectives of This PPT

We will discuss **Chapter One** of the **textbook** (Sommerville, 10th Ed.)

Objectives of This PPT

We will discuss **Chapter One** of the **textbook** (Sommerville, 10th Ed.) and supplement this with material from **Chapter Two** of the **2014 ACM-IEEE Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering** (CGUDP).

Objectives of This PPT

We will discuss **Chapter One** of the **textbook** (Sommerville, 10th Ed.) and supplement this with material from **Chapter Two** of the **2014 ACM-IEEE Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering** (CGUDP).
This includes the following:

- Defining Software Engineering (**SWE**),
- Providing a brief history of it's evolution,
- Outlining the ethical code of conduct underlying SWE.
- This will be covered in *three* PPTs.

Table of Contents

Today

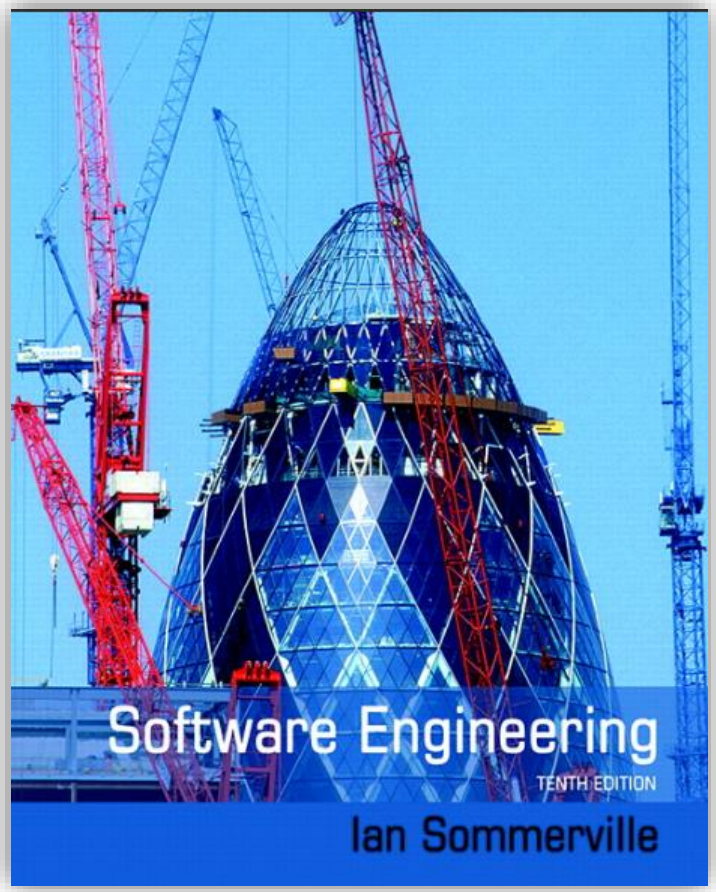


Table of Contents

Table of Contents

Objectives Sommerville, 9th Ed.

The objectives of this chapter are to introduce software engineering and to provide a framework for understanding the rest of the book. When you have read this chapter you will:

Today

- understand what software engineering is and why it is important;
- understand that the development of different types of software systems may require different software engineering techniques;
- understand some ethical and professional issues that are important for software engineers;
- have been introduced to three systems, of different types, that will be used as examples throughout the book.

Lectures for Weeks One and Two

WEEKS	TOPIC	CHAPTER
1	Administration and Introduction.	1
2	Continued	1
3	Introduction to Software Processes	2
4	Agile	3
		4
		Chapters 1 - 4 plus addendum materials
	sthem Modeling	5
		5
		6
	ation	7
	Verification	8
	d Maintenance	9
	for Final	Comprehensive

- 0 - Administration
- 1.1 - Introduction to Software Engineering 1
- 1.2 - Introduction to Software Engineering 2
- 1.3 - Introduction to Software Engineering 3
- 1.4A - Project Overview and SWE Tools
- 1.5A - Group Psychology and Teamwork
- 1.6A - UML - A Standardized Tool for Teams
- 1.7A - Software Project Management

Preamble


Projektmanagement
XML-Technologien
Web-Technologien
Konzeption
Soziale Kompetenz
Requirements Engineering
GUI
Methoden
Software Engineering
Business Engineering
Prozesse
Aktuelle Themen
Compiler
Tools
Architektur
Wissensbasierte Systeme
Quality-Management
Testen
Vorgehensmodelle
Systemprogrammierung
Programmierparadigmen
Wartung
Modelle
Analyse
Sichere Systeme
Prinzipien
Implementierung
Entwurf

Question: What are some of the largest programs?

Question: What are some of the largest programs?

- What do we mean by “largest”?

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?

Project	SLOC
Lucent 5ESS Switch	100m
Windows Vista	50m
Red Hat Linux 7.1	30m
Windows XP	40m
Visual Studio	40m
MS Office	30m

http://www.stackprinter.com/

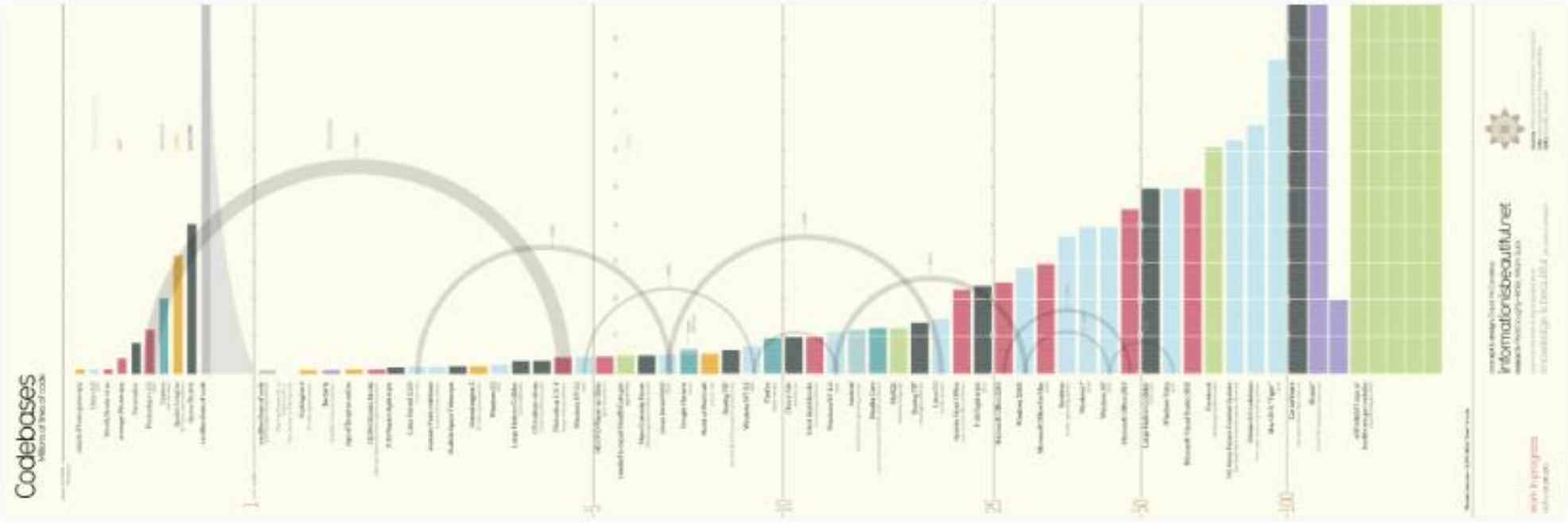
Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - It is estimated that that the software needed to run all of Google’s Internet services—from Google Search to Gmail to Google Maps—spans some **2 billion lines of code**.

https://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Apparently, the human genome project has **3.3 billion LOC**



https://codybonney.com/codebases-millions-of-lines-of-code-infographic/

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?

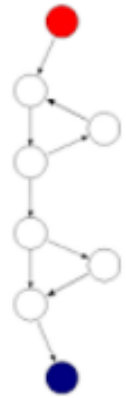
Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?
    - How do we measure complexity?

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?
    - How do we measure complexity?

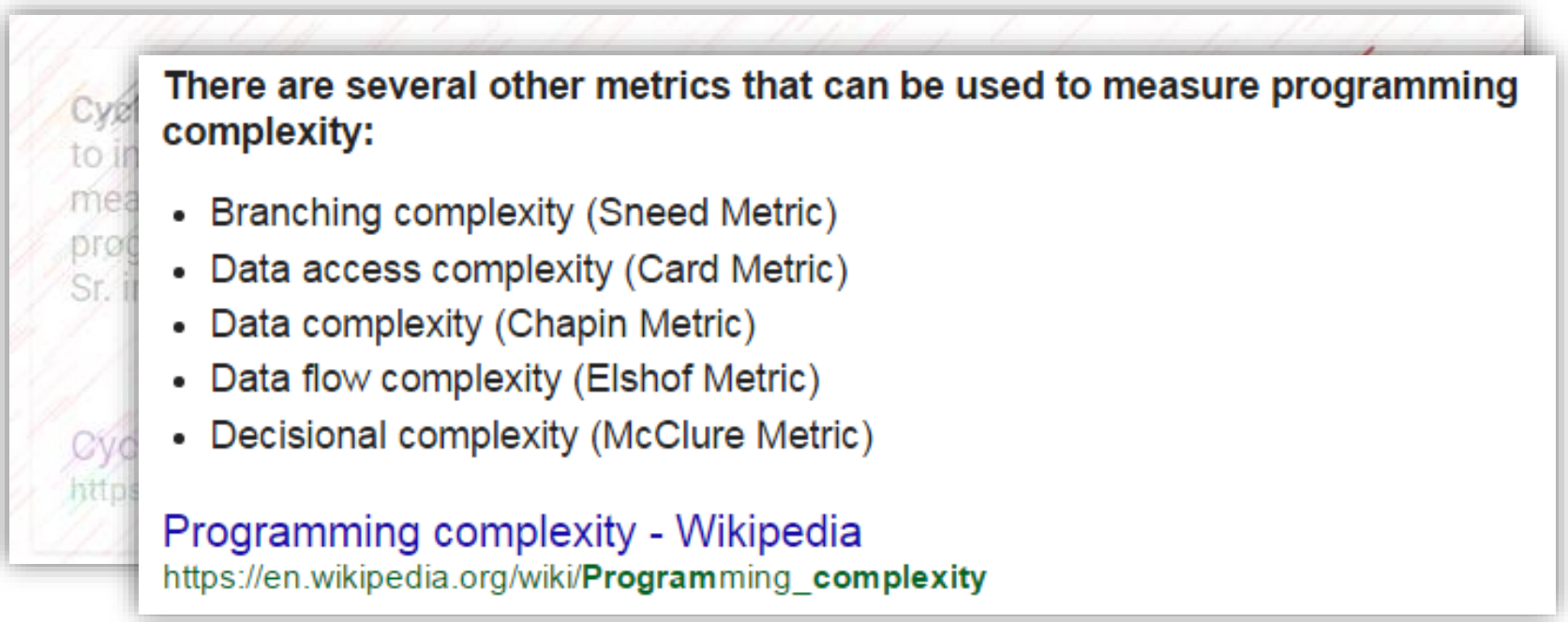
Cyclomatic Complexity is a software metric (measurement), used to indicate the **complexity** of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976.



Cyclomatic complexity - Wikipedia
https://en.wikipedia.org/wiki/**Cyclomatic_complexity**

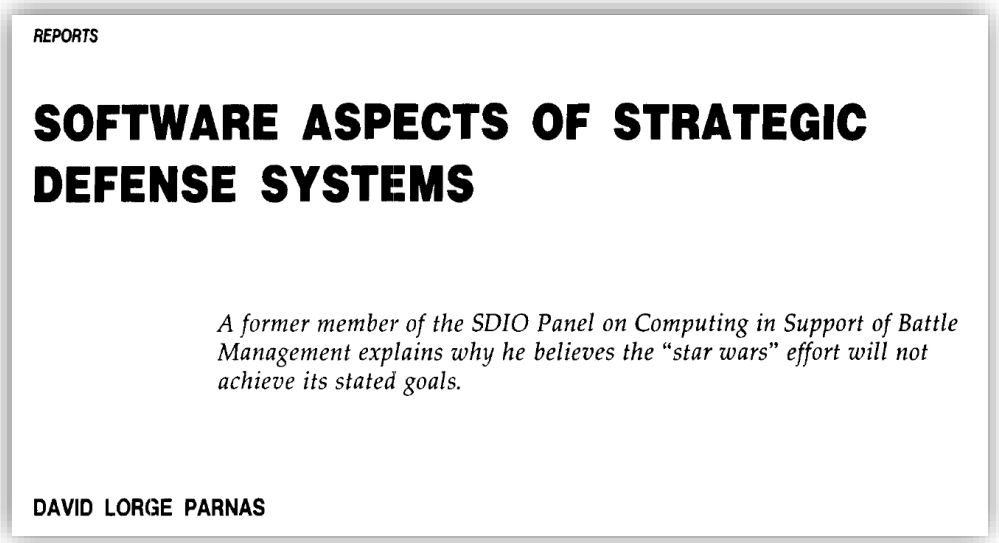
Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?
    - How do we measure complexity?



Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?
    - How do we measure complexity?
  - Large in terms of the time and effort and the number of people involved?



REPORTS

SOFTWARE ASPECTS OF STRATEGIC DEFENSE SYSTEMS

A former member of the SDIO Panel on Computing in Support of Battle Management explains why he believes the “star wars” effort will not achieve its stated goals.

DAVID LORGE PARNAS

Question: What are some of the largest programs?

- What do we mean by “largest”?
  - SLOC (Source Lines Of Code)?
  - Large in terms of Complexity?
    - How do we measure complexity?
  - Large in terms of the time and effort and the number of people involved?
  - Large in terms of cost?

Started	Terminated	System name	Type of system	Country or region	Type of purchaser	Problems	Cost *(expected)*	Outsourced or in-house?	Outcome
1980s	1993	TAURUS	Electronic trading platform	United Kingdom (London)	Stock exchange	Scope creep, cost overrun. The project was never completed.	£75m	?	Cancelled
1984	1990	RISP	Integrated computer services	United Kingdom (Wessex)	Wessex Health Authority	Scope creep, cost overrun. The project was never completed.	£63m *(£29m)*	?	Cancelled
1997	2000	Bolit	Customer service, finance and administration system	Sweden	Patent and Registration Office	Too complicated, bad functioning, cost overrun. The project was after completion never used, the agency still today does not have a working IT system.[1][2]	SEK 300m ($35m)	Outsourced	Scrapped
2002	2011	NHS Connecting for Health	Electronic care records	United Kingdom	Central government	Beset by delays and ballooning costs, and the software part of it was never finished. The government was also criticised for not demonstrating value for money. Although the contracts were drafted to ensure that the contractors would be forced to bear a significant portion of the cost of the project going wrong if it did go wrong, in reality this did not always happen. The NPfIT was described by Members of Parliament as one of the "worst and most expensive contracting fiascos" ever.[3]	£12bn *(£2.3bn)*	Outsourced	Discontinued, but some parts continued
2005	2012	Expeditionary Combat Support System	Military Enterprise Resource Planning	United States	Air force	No significant capabilities ready on time; would have cost $1.1bn more just to get to 1/4 of the original scope.	$1.1bn	Outsourced - including requirements	Cancelled
2007	2012	da:Polsag	Police case management	Denmark	Police	Did not work properly, technical problems with contractor.	DKK 500m ($70m)	Outsourced	Cancelled

https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects

Question: What are some of the worst software failures?

Question: What are some of the worst software failures?

A Collection of Well-Known Software Failures

Software systems are pervasive in all aspects of society. From electronic voting to online shopping, a significant part of our daily life is mediated by software. In this page, I collect a list of well-known software failures. I will start with a study of economic cost of software bugs.

Table of contents

The number of *s is the ironic factor I assign to each story. The one with most *s is the most ironic one.

http://www.cse.psu.edu/~gxt29/bug/softwarebug.html

Question: What are some of the worst software failures?

Economic Cost of Software Bugs

Report Date: 2/2002 Price Tag: $60 Billion Annually

WASHINGTON (COMPUTERWORLD) - Software bugs are costing the U.S. economy an estimated $59.5 billion each year; with more than half of the cost borne by end users and the remainder by developers and vendors, according to a new federal study.

Improvements in testing could reduce this cost by about a third, or $22.5 billion, but it won't eliminate all software errors, the study said. Of the total $59.5 billion cost, users incurred 64% of the cost and developers 36%.

http://www.cse.psu.edu/~gxt29/bug/softwarebug.html

Notable S/W Failures for 2017 according to the Validata Group

- **Suncorp Bank – Vanishing cash**

  In February of this year, a malfunction during a routine upgrade caused the disappearance of money from customers' bank accounts. Additional customer complaints included overdrawn and locked out accounts.

- **Bitcoin – Unlimited node crash**

  Bitcoin suffered from two software failures in the same month back in March. The most serious glitch was linked to a software bug that caused over 100 Bitcoin Unlimited nodes (Nearly 70% of the nodes running Bitcoin Unlimited at the time) to disappear from the network completely.

- **Dodge Ram – 1.25 million recalls**

  A major software glitch that could cause the airbags and seatbelts in Ram trucks to fail during rollover collisions caused Dodge to recall more that 1.25 million trucks. To prevent the problem, the FCA must now reprogram the onboard sensor of every impacted vehicle.

- **Cairns Hospital – Security patch gone wrong**

  A catastrophic glitch affecting five Australian hospitals was introduced during the application of security patches designed to counter potential future cyber attacks. It required more than two weeks for the hospitals to recover their electronic medical record systems.

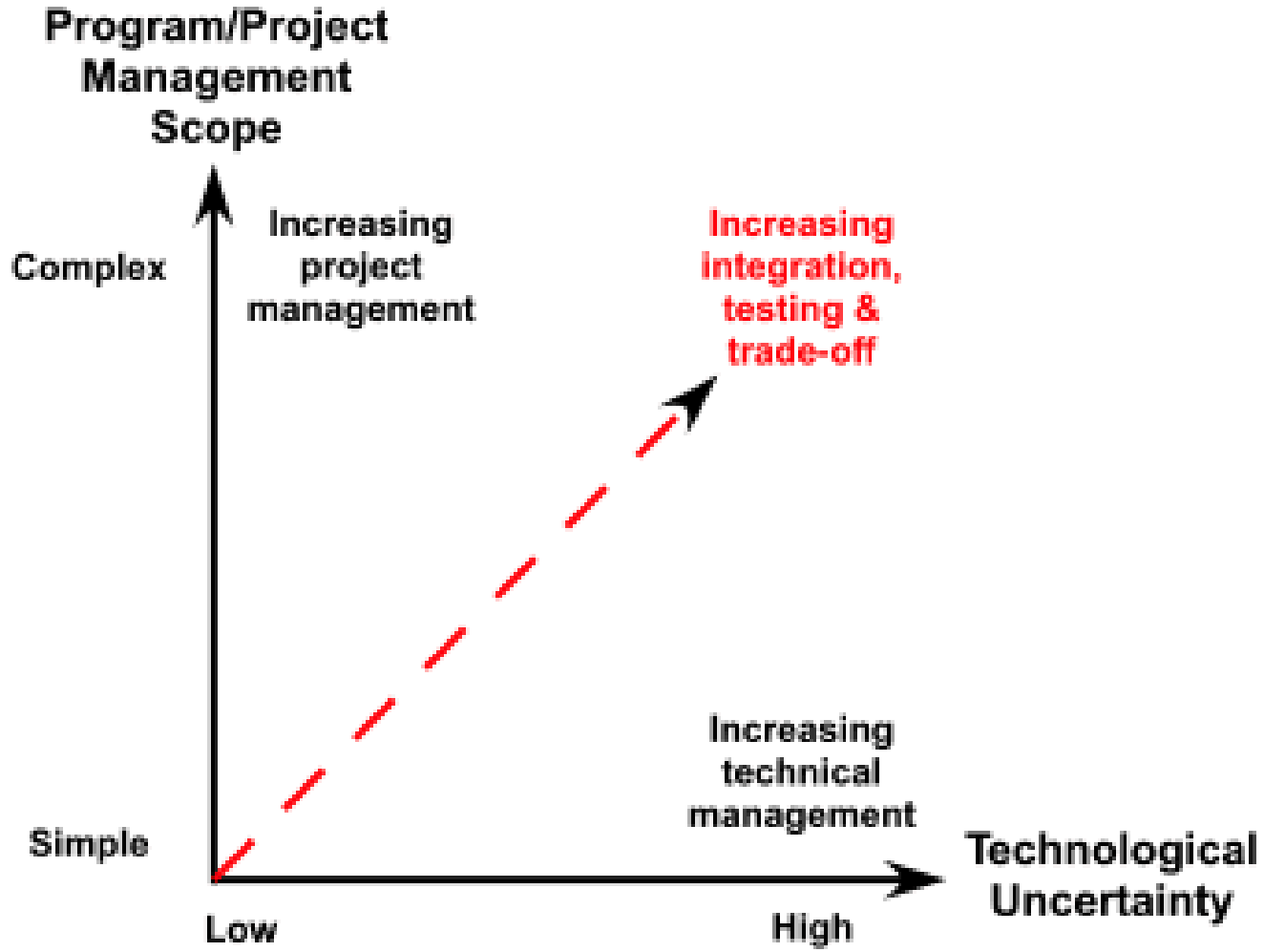
- **British Airways – Disruption for 75,000 passengers**

  For the *sixth time* this year – a major IT software failure led to massive cancellations on local flights and significant delays on international flights. According to NPR.org – it took over three days of cancellation chaos to resolve the problems that plagued BA during this outage.

Compare all this to Building Bridges

How the Tallest Bridge on Earth Was Built

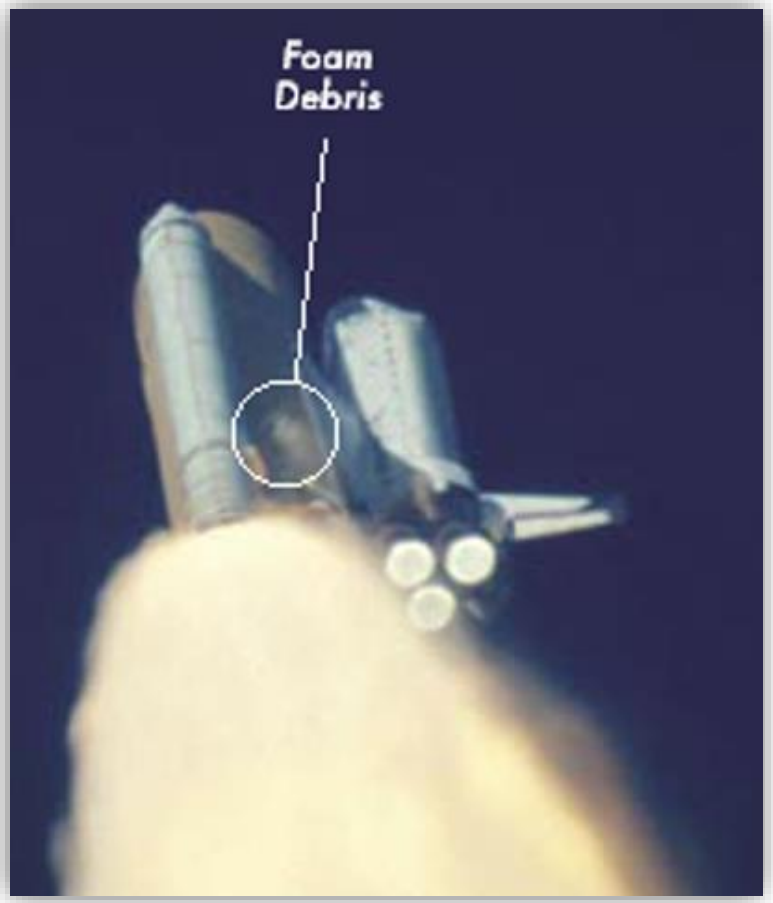
There are Two Dimensions to Any Large-Scale Project.



http://www.maxwideman.com/papers/success/nature.htm

Two Dimensions: Administrative and Technical

System failures are often due to **administrative failure** as well as **technical malfunctions** due to design failures.



http://www.cse.psu.edu/~gxt29/bug/softwarebug.html

Question: What is the most sophisticated malware?

mal·ware

/ˈmalwer/

noun COMPUTING

software that is intended to damage or disable computers and computer systems.

Question: What is the most sophisticated malware?

What are the Reasons for S/W Bugs?

Ten Reasons for S/W Bugs:

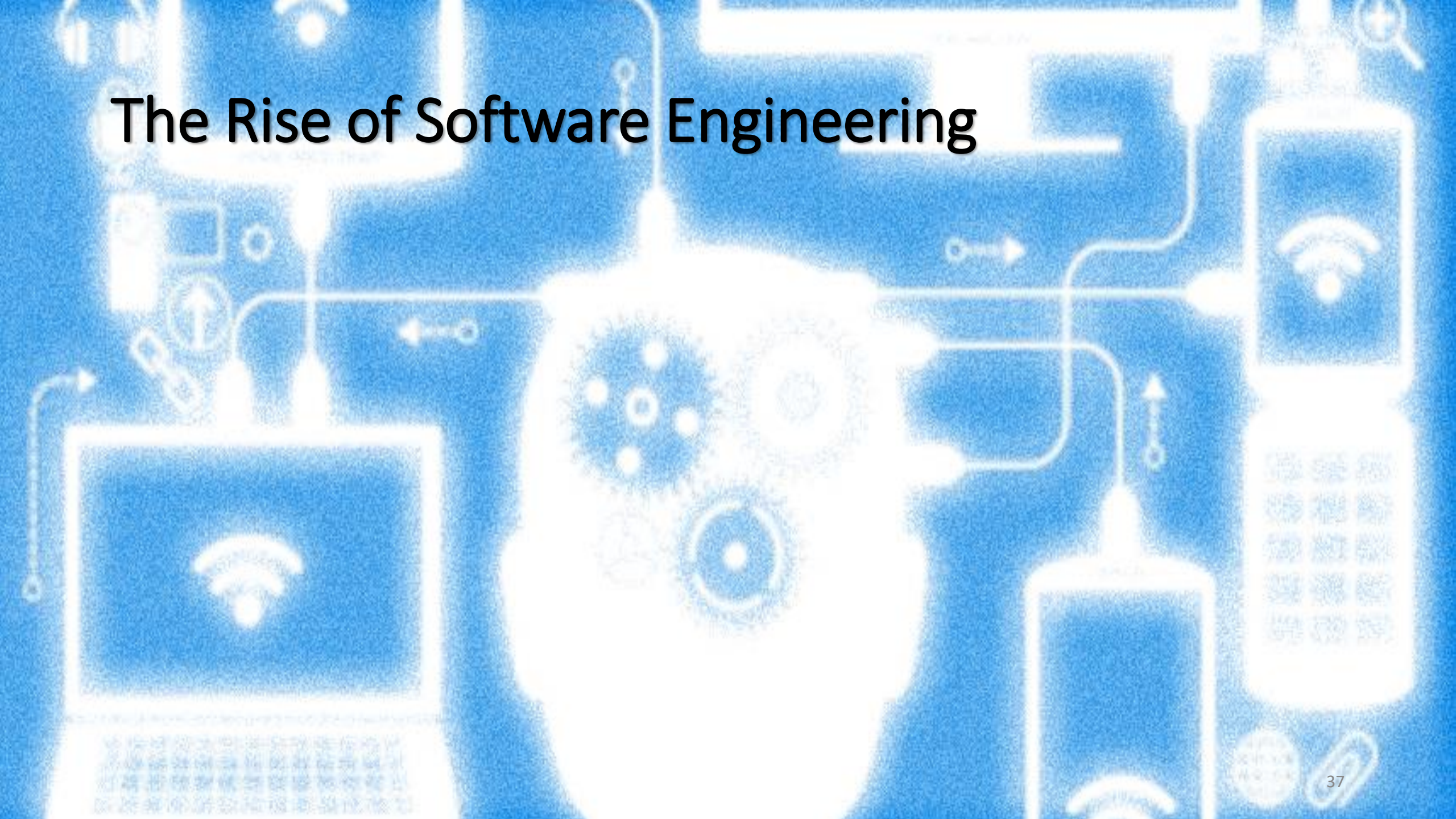
1. Miscommunication or no communication
2. Software complexity
3. Programming errors
4. Changing requirements
5. Time pressures
6. Egotistical or overconfident people (CMPT 275 – Spring, 2001)
7. Poorly documented code
8. Software development tools
9. Obsolete automation scripts
10. Lack of skilled testers (lack of extraordinary people)

http://www.softwaretestinghelp.com/why-does-software-have-bugs/

https://ifthenwow.com/blog/category/technology/

END PREAMBLE

The Rise of Software Engineering



The Rise of Software Engineering

- Since the dawn of electronic computing in the 1940s, and the commercialization of computers in the mid-1950s, computing systems and their applications have evolved at a staggering rate.
- Software plays a central role in almost all aspects of daily life: communications, government, manufacturing, banking and finance, education, transportation, entertainment, medicine, agriculture, and law.
- The number, size, and application domains of computer programs have grown dramatically and, as a result, huge sums of money are being spent on software development.
- Most people's lives and livelihoods depend on this development's effectiveness.
- Software products help us to be more efficient and productive. They provide information, make us more effective problem solvers, and provide us with safer, more flexible, and less confining work, entertainment, and recreation environments.

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

- Since the dawn of electronic computing in the 1940s, and the commercialization of computers in the mid-1950s, computing systems and their applications have evolved at a staggering rate.
- Software plays a central role in almost all aspects of daily life: communications, government, manufacturing, banking and finance, education, transportation, entertainment, medicine, agriculture, and law.
- The number, size, and application domains of computer programs have grown dramatically and, as a result, huge sums of money are being spent on software development.
- Most people's lives and livelihoods depend on this development's effectiveness.
- Software products help us to be more efficient and productive. They provide information, make us more effective problem solvers, and provide us with safer, more flexible, and less confining work, entertainment, and recreation environments.
- **It Begs The Question**: Is there a "Moore's Law" for software?
  - See Wirth's law, and "Software Bloat".

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

- Software products are among the most **complex** manmade systems, and by its very nature, software has intrinsic, essential properties (for example, complexity, invisibility, and changeability) that are not easily addressed [Brooks 1987].

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

- Software products are among the most complex manmade systems, and by its very nature, software has intrinsic, essential properties (for example, complexity, invisibility, and changeability) that are not easily addressed [Brooks 1987].
- Programming techniques and processes that work effectively when used by an individual or small team to develop modest-sized programs **do not scale well** to the development of large, complex systems. (Complexity can arise with just a few hundred lines of code, and large systems can run to millions of lines of code, requiring years of work by hundreds of software developers.)

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

- Software products are among the most complex manmade systems, and by its very nature, software has intrinsic, essential properties (for example, complexity, invisibility, and changeability) that are not easily addressed [Brooks 1987].
- Programming techniques and processes that work effectively when used by an individual or small team to develop modest-sized programs do not scale well to the development of large, complex systems. (Complexity can arise with just a few hundred lines of code, and large systems can run to millions of lines of code, requiring years of work by hundreds of software developers.)
- The **pace of change** in computer and software technology drives the demand for new and evolved software products. This situation has created customer expectations and competitive forces that strain our ability to produce quality software within acceptable development schedules.

https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

- Software products are among the most complex manmade systems, and by its very nature, software has intrinsic, essential properties (for example, complexity, invisibility, and changeability) that are not easily addressed [Brooks 1987].
- Programming techniques and processes that work effectively when used by an individual or small team to develop modest-sized programs do not scale well to the development of large, complex systems. (Complexity can arise with just a few hundred lines of code, and large systems can run to millions of lines of code, requiring years of work by hundreds of software developers.)
- The pace of change in computer and software technology drives the demand for new and evolved software products. This situation has created customer expectations and competitive forces that strain our ability to produce quality software within acceptable development schedules.
- The **availability of qualified software engineers** has not kept pace with the demand from industry, so that systems are designed and built by people with insufficient educational background or experience.

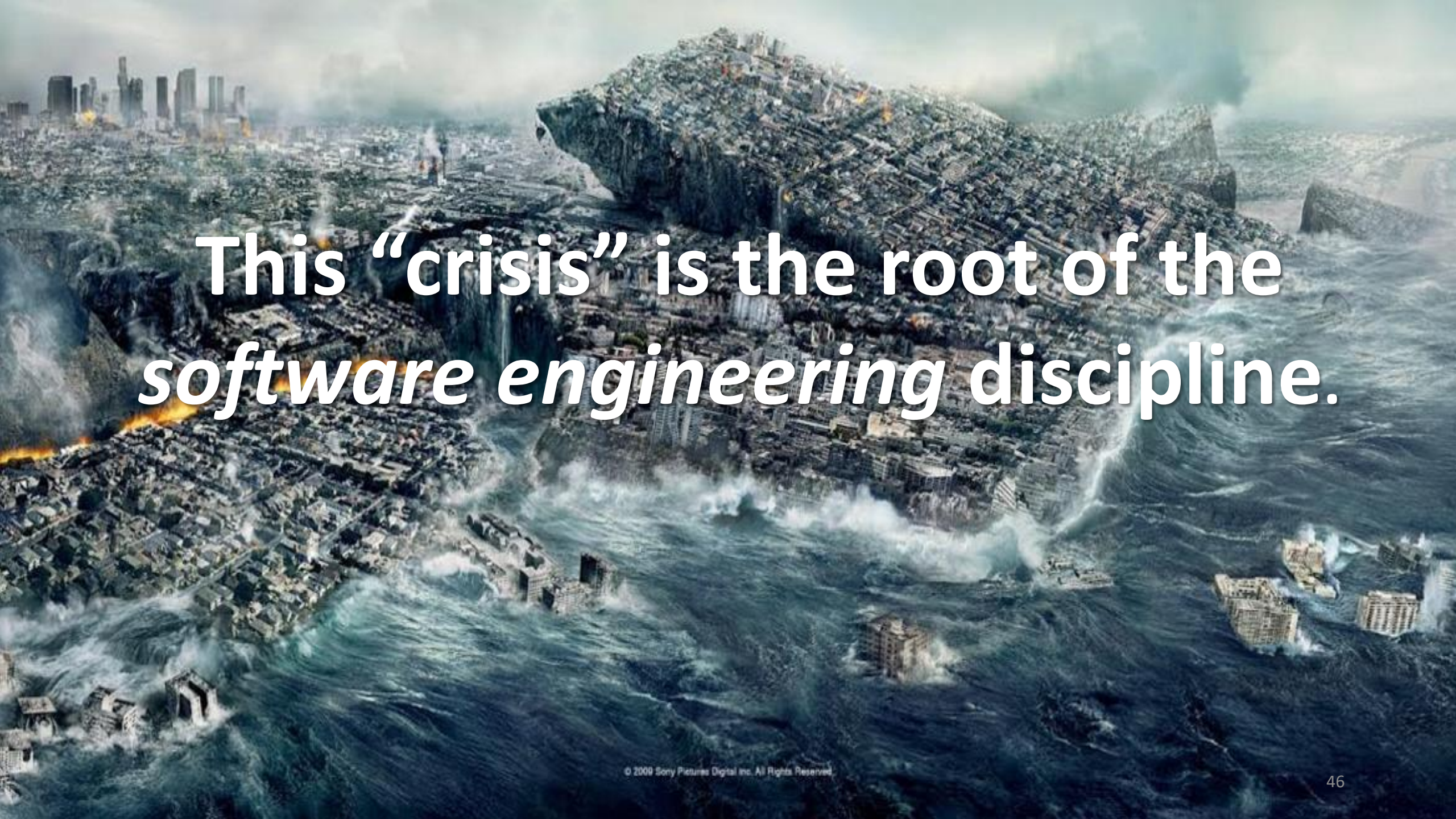
https://www.acm.org/education/se2014.pdf

The Rise of Software Engineering

Despite these successes, this period has witnessed **serious problems** in terms of the development costs, timeliness, and quality of many software products. There are many reasons for these problems:

- Software products are among the most complex manmade systems, and by its very nature, software has intrinsic, essential properties (for example, complexity, invisibility, and changeability) that are not easily addressed [Brooks 1987].
- Programming techniques and processes that work effectively when used by an individual or small team to develop modest-sized programs do not scale well to the development of large, complex systems. (Complexity can arise with just a few hundred lines of code, and large systems can run to millions of lines of code, requiring years of work by hundreds of software developers.)
- The pace of change in computer and software technology drives the demand for new and evolved software products. This situation has created customer expectations and competitive forces that strain our ability to produce quality software within acceptable development schedules.
- The **availability of qualified software engineers** has not kept pace with the demand from industry, so that systems are designed and built by people with insufficient educational background or experience.
- This has led to the so-called **Software Crisis**.

https://www.acm.org/education/se2014.pdf


This “crisis” is the root of the *software engineering* discipline.
© 2009 Sony Pictures Digital Inc. All Rights Reserved.

The Rise of Software Engineering

History of software engineering

The notion of 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called the 'software crisis' (Naur and Randell, 1969). It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late.

Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding and object-oriented development. Tools and standard notations were developed and are now extensively used.

http://www.SoftwareEngineering-9.com/Web/History/

Sommerville, 9th Ed.

The Rise of Software Engineering

NOTE: Please refer to section 2.2, **Software Engineering 2014: Curriculum Guidelines for Undergraduate for a detailed history of SWE** (*https://www.acm.org/education/se2014.pdf*).

The Rise of Software Engineering - Milestones

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

https://en.wikipedia.org/wiki/Software_development_process

The Rise of Software Engineering - Milestones

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

Structured programming is a **programming** paradigm aimed at improving the clarity, quality, and development time of a computer **program** by making extensive use of subroutines, block structures, for and while loops—in contrast to using simple tests and jumps such as the go to statement which could lead to "spaghetti code" ...

Structured programming - Wikipedia
https://en.wikipedia.org/wiki/Structured_programming

https://en.wikipedia.org/wiki/Software_development_process

The Rise of Software Engineering - Milestones

1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Dynamic systems development method (DSDM), since 1994
- Scrum, since 1995
- Team software process, since 1998
- Rational Unified Process (RUP), maintained by IBM since 1998
- Extreme programming, since 1999

2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
- Disciplined agile delivery (DAD) Supersedes AUP

https://en.wikipedia.org/wiki/Software_development_process

The Rise of Software Engineering - Milestones

1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s

Each form of OOP has its strengths and its weaknesses; its advocates and its detractors; its domains of utility and its domains of uselessness. There's nothing magical about OOP that makes it the Killer Paradigm and there's nothing infernal about it that makes it the Killer (of Programmers) Paradigm.

I can't really point you to any books or articles that killed my interest in OOP as a Silver Bullet (as opposed to one of many techniques I can use to keep my projects survivable). I can point to the funniest critique of a specific brand of OOP, however: Steve Yegge's classic "Execution in the Kingdom of Nouns".

- Disciplined agile delivery (DAD) Supersedes AUP

stackoverflow

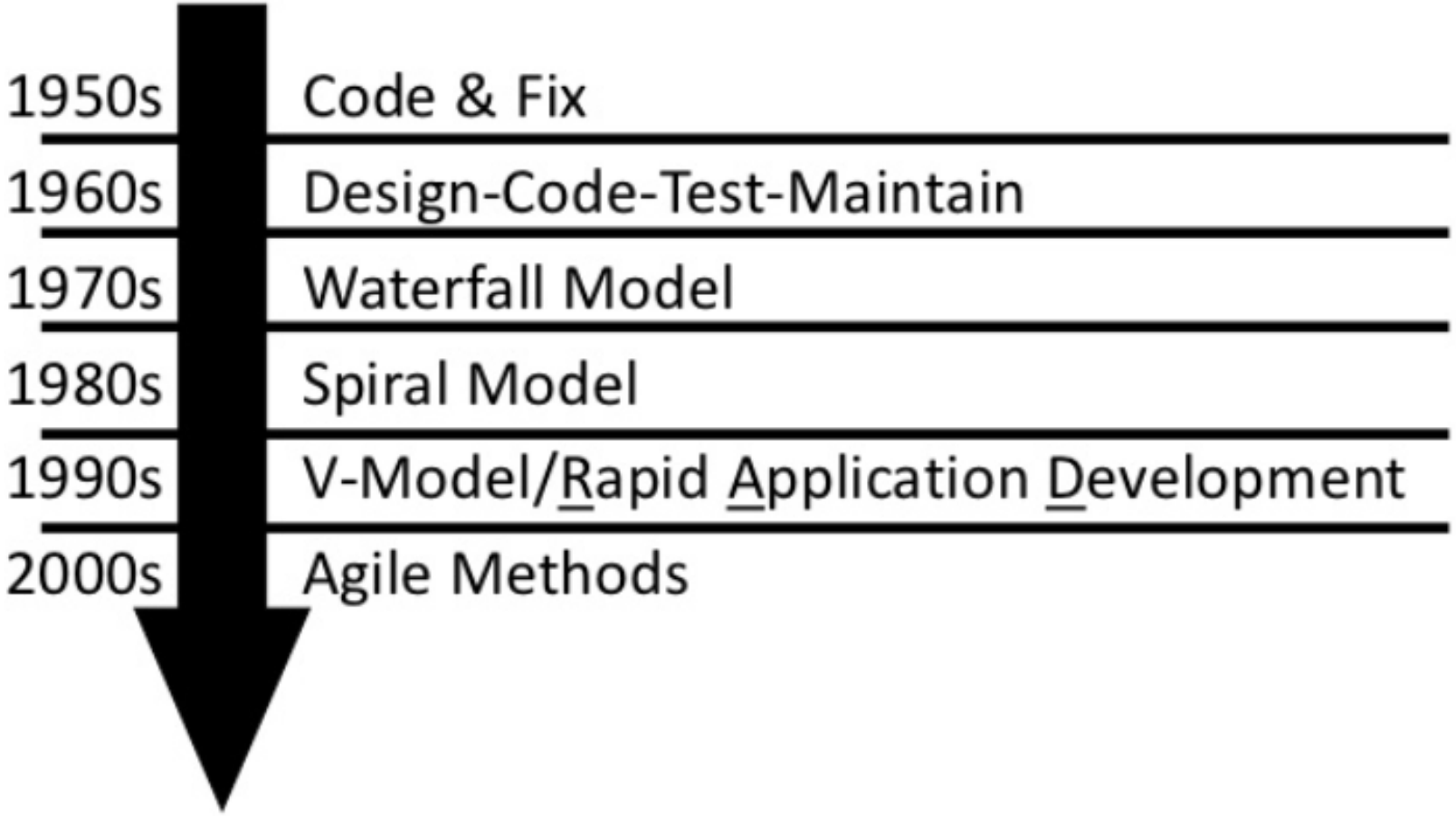
The Rise of Software Engineering - Milestones

2010s

- Scaled Agile Framework (SAFe)
- Large-Scale-Scrum (LeSS)

https://en.wikipedia.org/wiki/Software_development_process

Summary of SWE Methodology Timeline



https://www.slideshare.net/DamianGordon1/software-engineering-methodologies

The Rise of Software Engineering - Caveat

Why can't I find well elaborated criticism on agile/scrum on the web?

6

After several years of agile and scrum hype around, I'd expect the first well-known reactions from "bullshit!" to "magical!" to be gone and I should be able to find some really elaborated articles that question the highly praised benefits of those modern management techniques. But I don't.

Aren't there any? Is scrum really *that* great? Is there a conspiracy going on?

Project Management beta

The Rise of Software Engineering - Warning

Why can't I find well elaborated criticism on agile/scrum on the web?

6

After several years of agile and scrum hype around, I'd expect the first well-known reactions from "bullshit!" to "magical!" to be gone and I should be able to find some really elaborated articles that question the highly praised benefits of those modern management techniques. But I don't.

Aren't there any? Is scrum really *that* great? Is there a conspiracy going on?

No, scrum isn't that "great." It is just another framework. It may or may not work for you.

But the reason I think it hasn't become a "Mac vs PC" war is the agile community knows that. There are few people in the agile community that will say agile is the one and only way. Most of those don't tend to stick around long as it pretty much flies in the face of the tenets of agile.

Much of the agile community is in the camp of "give it a try" and not "you must convert." A low barrier to entry, and a generally welcoming community means less conflict.

Me personally, I've always maintained it is just one of the many tools in my tool box. I find I reach for it more and more, but it still sits in the same box as the PMBoK, Conflict Management training, Manager-Tools, Customer Service Training and other tools.

Project Management beta

The Rise of Software Engineering - Warning

StackExchange | Search on User Experience Meta...

UX User Experience META | Questions | Tags

Feedback on Nielsen Norman Group's Usability Week 2012

3

I am curious to know if **anyone has been or knows about the Usability Week conferences that Nielsen Norman Group has organized for 2012 and what is their impression. Is it quality training or can achieve the same by reading some books? Is it overpriced or good value for money? Which are the most recommended conferences?**

I've got the feeling that training in new trends such as Agile or UX is highly overpriced —almost scammy— but on the other hand NN/g is a reputable institution.

Preliminary Conclusion

- SWE is “trendy”.
- Real Engineering is about physical objects – where is the “engineering” in SWE?
- What is “scientific” about SWE processes considering that Engineering is about “Applied Science”?
- Specifically, is SWE more about business (management) or applied science (*a.k.a.,* engineering)?

Defining Software Engineering

First: Let Us Define the Large-Scale Setting

DEFINITION: Computing Science

DEFINITION: Computing Science

The study of *algorithms*,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,
- Their linguistic realizations,

DEFINITION: Computing Science

The study of *algorithms*, including

- Their formal and mathematical properties,
- Their hardware realizations,
- Their linguistic realizations,
- Their applications.

- Schneider and Gersting, 1999

Definition: Algorithm

Definition: Algorithm

A well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and *solves a problem* in a finite amount of time.

- Schneider and Gersting, 1999 (modified by Pearce)

Areas of Computing Science

Areas of Computing Science

- Data Structures (Efficient Organization of Data)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)
- Information Storage and Retrieval (DBMS)

Areas of Computing Science

- Data Structures (Efficient Organization of Data)
- Operating Systems (Managing the Computer)
- Programming Languages (Software)
- Computational Complexity (Limitations of Computing)
- Architecture (Design of Computers)
- Intelligent Systems (AI)
- Automata Theory (Abstract Computers)
- Information Storage and Retrieval (DBMS)
- Software Engineering (Large Systems)

Defining Software Engineering

Defining Software Engineering

What exactly is software?

Defining Software Engineering

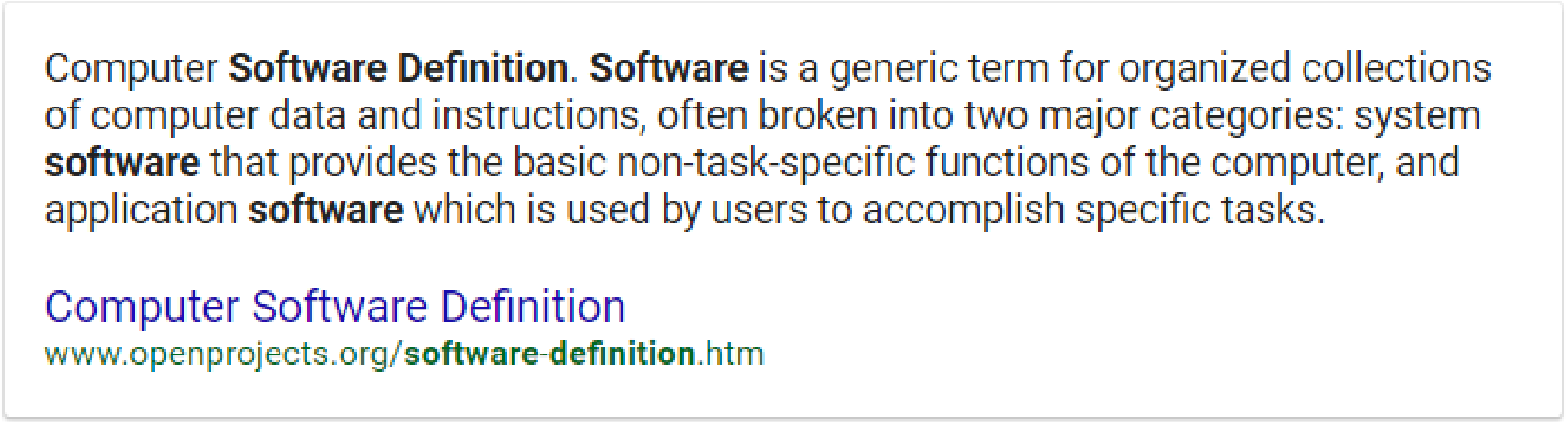
What exactly is software?

Computer **algorithms** solve the problem but computer **programs** implement them. A computer **program** is sequence of instruction that comply the rules of specific programming language written to perform a specified task with a computer.

What is the difference between an algorithm and a computer program ...
https://www.quora.com/What-is-the-**difference-between-an-algorithm**-and-a-computer-**p**...

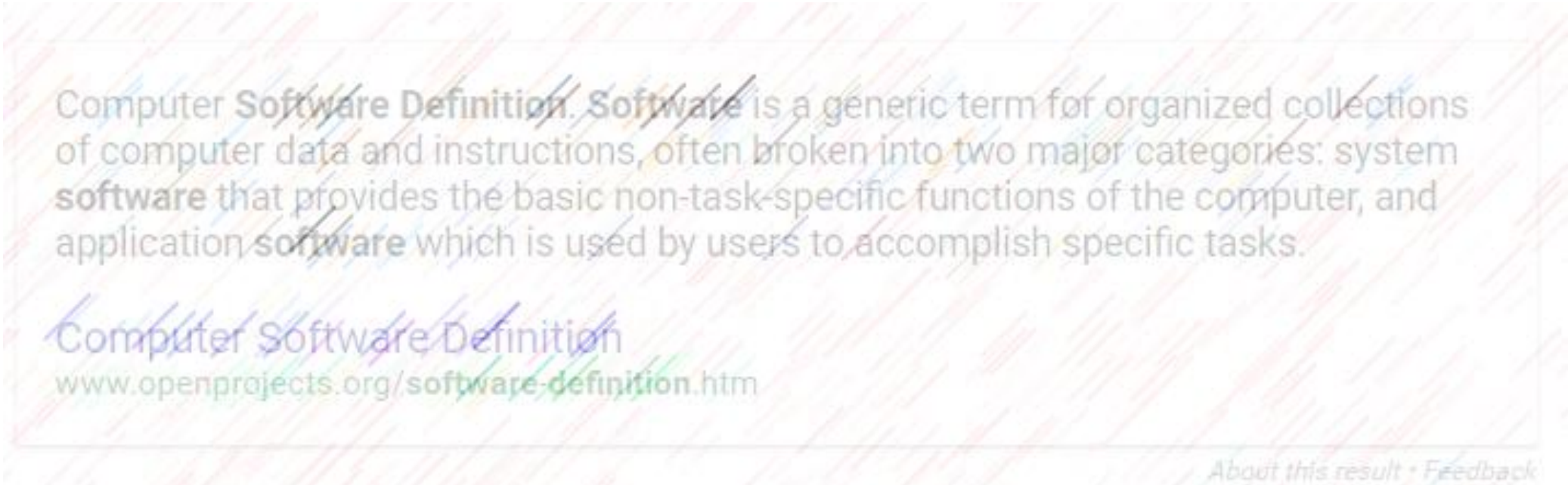
Defining Software Engineering

What exactly is software?



Defining Software Engineering

What exactly is software?



The term "software" was first proposed by Alan Turing and used in this sense by John W. Tukey in 1957. In computer science and software engineering, computer software is all information processed by computer systems, programs and data. -*Wikipedia*

Defining Software Engineering

What exactly is engineering?

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

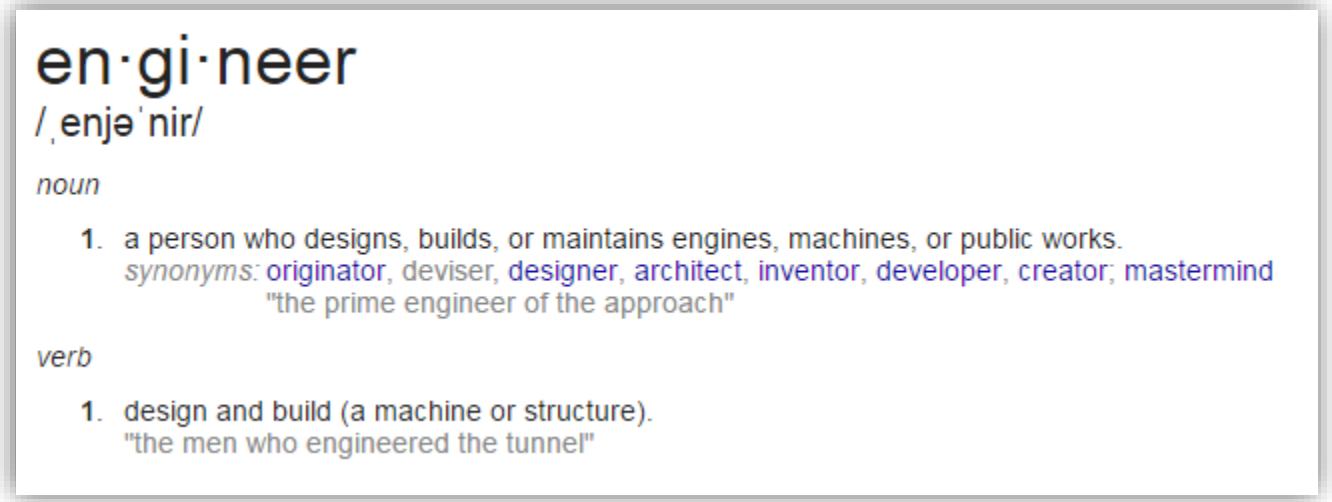
1 : the activities or function of an engineer

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

1 : the activities or function of an engineer



en·gi·neer
/ˌenjəˈnir/

noun

1. a person who designs, builds, or maintains engines, machines, or public works.
   synonyms: originator, deviser, designer, architect, inventor, developer, creator; mastermind
   "the prime engineer of the approach"

verb

1. design and build (a machine or structure).
   "the men who engineered the tunnel"

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

1 : the activities or function of an engineer

2 **a** : the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

1 : the activities or function of an engineer

2 **a** : the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

b : the design and manufacture of complex products <software *engineering*>

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

1 : the activities or function of an engineer

2 **a :** the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people
b : the design and manufacture of complex products <software *engineering*>

3 : calculated manipulation or direction (as of behavior) <social *engineering*> — compare GENETIC ENGINEERING

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

- The term *Engineering* is derived from the Latin *ingenium*, meaning "cleverness" and *ingeniare*, meaning "to contrive, devise".
- The word "engine" derives from Old French *engin*, from the Latin *ingenium*–the root of the word *ingenious*.

WIKIPEDIA
The Free Encyclopedia

Defining Software Engineering

What exactly is engineering?

Definition of ENGINEERING

- The term *Engineering* is derived from the Latin *ingenium*, meaning "cleverness" and *ingeniare*, meaning "to contrive, devise".
- The word "engine" derives from Old French *engin*, from the Latin *ingenium*–the root of the word *ingenious*.

Making ENGINES

Defining Software Engineering

NOTE: What exactly is TECHNOLOGY?

noun. /tɛkˈnɑlədʒi/ (pl. **technologies**) 1[uncountable, countable] scientific knowledge used in practical ways in industry, for example in designing new machines science and **technology** recent advances in medical **technology** to make use of the most modern **technologies** see also high **technology**, information **technology**.

technology noun - Oxford Learner's Dictionaries
https://www.oxfordlearnersdictionaries.com/definition/american_english/technology

The use of scientific knowledge for practical purposes – **Applied Science**.

Defining Software Engineering

What is Software Engineering?

Defining Software Engineering

Software engineering (**SWE**) is the application of engineering to the design, development, implementation, testing and maintenance of software in a systematic method.[1][2][3]

Typical formal definitions of **software engineering** are:

- "research, design, develop, and test operating systems-level software, compilers, and network distribution software for medical, industrial, military, communications, aerospace, business, scientific, and general computing applications."[4]
- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software";[5]
- "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software";[6]
- "an engineering discipline that is concerned with all aspects of software production";[7]
- and "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines."[8]

WIKIPEDIA
The Free Encyclopedia

Defining Software Engineering

Software engineering (**SWE**) is the application of engineering to the design, development, implementation, testing and maintenance of software in a systematic method.[1][2][3]

Typical formal definitions of **software engineering** are:

- "research, design, develop, and test operating systems-level software, compilers, and network distribution software for medical, industrial, military, communications, aerospace, business, scientific, and general computing applications";[4]
- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software";[5]
- "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software";[6]
- "an engineering discipline that is concerned with all aspects of software production";[7]
- and "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines".[8]

You wish

Defining Software Engineering - IEEE

The IEEE's 2010 definition states that software engineering is

> The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. [IEEE 2010]

Defining Software Engineering - IEEE

The IEEE's 2010 definition states that software engineering is

> The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. [IEEE 2010]

The nature of software, however, complicates the interpretation of these words [Brooks 1987]. To address this ambiguity, it is helpful to identify some key characteristics of software and the associated challenges that they create for any form of "engineering" process.

Defining Software Engineering

- Software is *abstract* and *invisible*. These characteristics present challenges for managing software development because of the problems they create for important engineering concepts such as quantification and measurement. They also complicate efforts to describe and organize software in ways that will facilitate knowledge exchange during the processes of its design, implementation, and maintenance.

Defining Software Engineering

- Software is *abstract* and *invisible*.
- Software has both *static* and *dynamic* properties. This duality provides a further challenge for description and measurement. It also makes it difficult to predict the effects arising from any changes made to a software product.

Defining Software Engineering

- Software is *abstract* and *invisible*.
- Software has both *static* and *dynamic* properties.
- Software is *intrinsically complex* in terms of its organization. Even a small software unit may possess many different execution paths, and there may be a large and complex set of relationships among its elements. This in turn presents challenges for verification and validation, documentation, and maintenance.

Defining Software Engineering

- Software is *abstract* and *invisible*.
- Software has both *static* and *dynamic* properties.
- Software is *intrinsically complex* in terms of its organization.
- No universal measures of *quality* exist for assessing a software product [Hughes 2000]. An engineering process should lead to products that are of “good” quality, but the relative importance of different quality measures will vary with the role of the product and differ for each “stakeholder” (producer, customer, or user).

Defining Software Engineering

- Software is *abstract* and *invisible*.
- Software has both *static* and *dynamic* properties.
- Software is *intrinsically complex* in terms of its organization.
- No universal measures of *quality* exist for assessing a software product [Hughes 2000].
- The *manufacturing* cycle for software products is not a significant element in software development, and it mainly involves the needs of distribution mechanisms. Software development is essentially a process that involves a progression through many layers of design abstraction and, hence, is unlike any conventional engineering processes, such as those that occur within mechanical and civil engineering.

Defining Software Engineering

- Software is *abstract* and *invisible*.
- Software has both *static* and *dynamic* properties.
- Software is *intrinsically complex* in terms of its organization.
- No universal measures of *quality* exist for assessing a software product [Hughes 2000].
- The *manufacturing* cycle for software products is not a significant element in software development, and it mainly involves the needs of distribution mechanisms.
- Software does *not wear out*. The maintenance activities associated with a software product are really part of an evolutionary design process.

Hardware Life Cycle

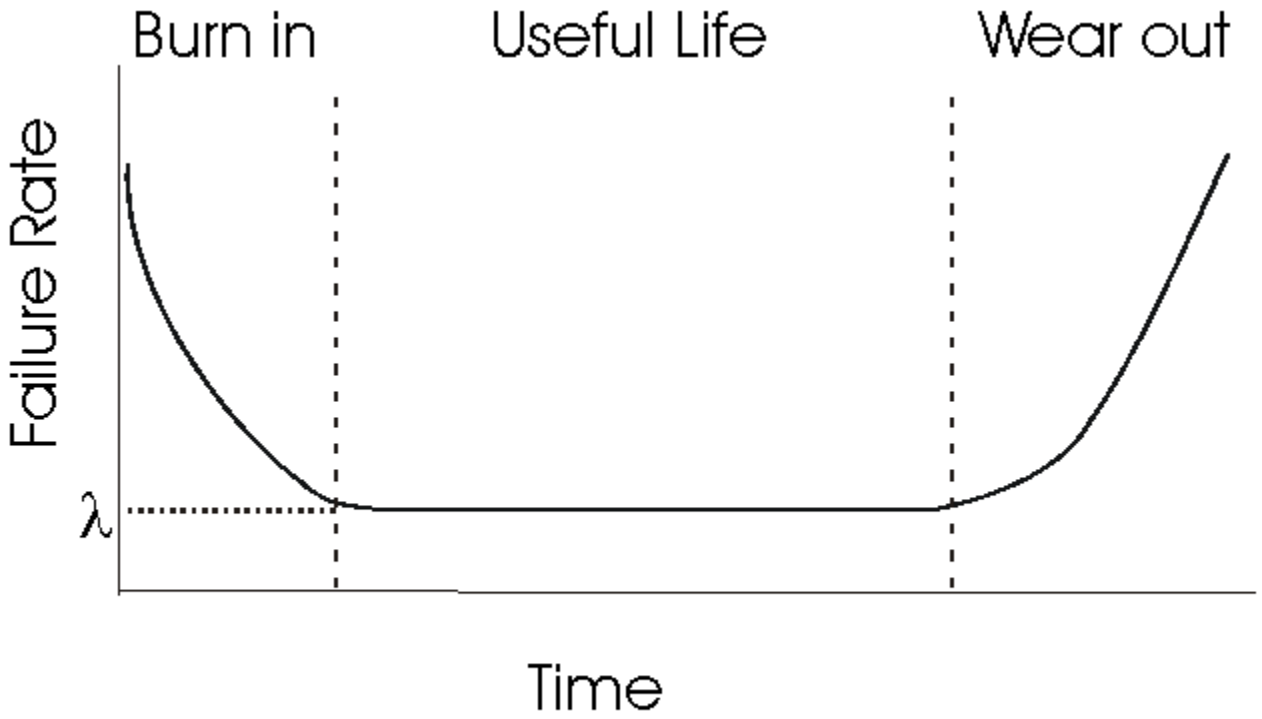


Figure 1. Bathtub curve for hardware reliability

http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

Software Life Cycle

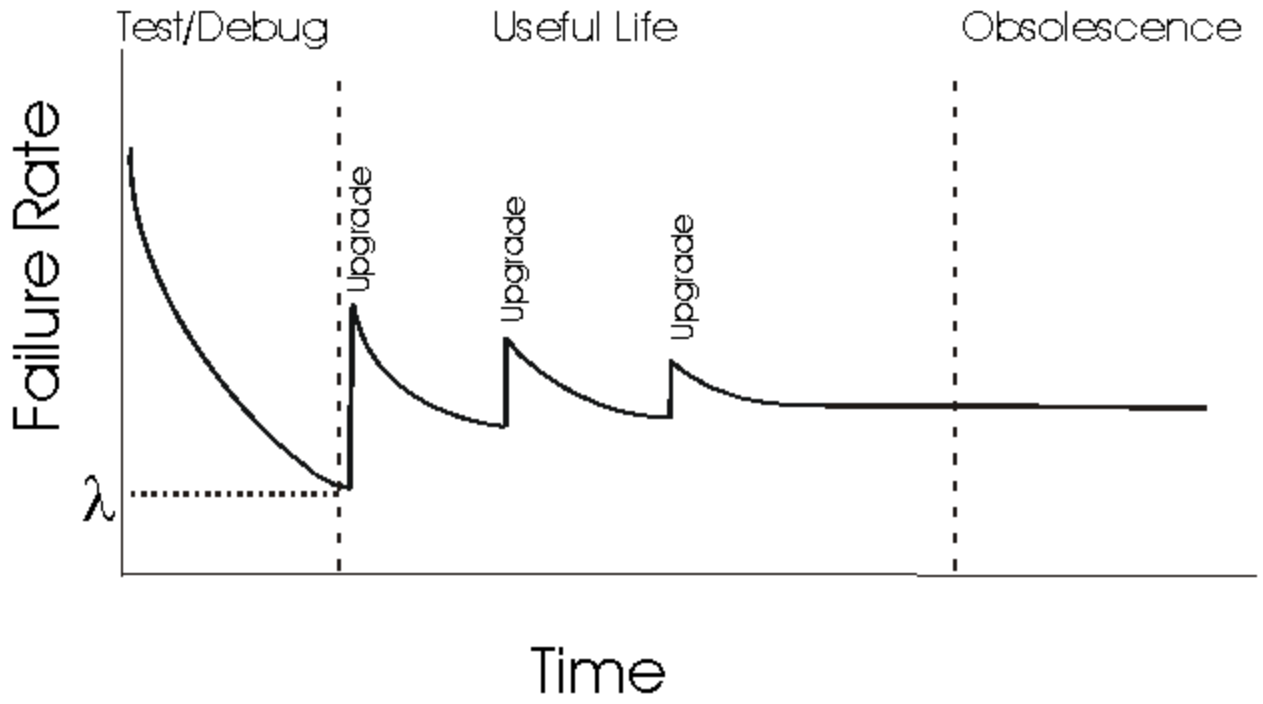


Figure 2. Revised bathtub curve for software reliability

http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

The Ambiguity of “Software Engineering”

1. Software projects are based on **logical** work, while the traditional notion of engineering is based on **physical** work.
2. Progress in software projects is largely **invisible** due largely to the 1st item, meaning customers of software projects cannot see the outcome in the middle of the project, since customers are not experts in coding and other technical work (an incomplete project will [only give a limited] outcome).
3. The **complexity** of a software project is difficult to estimate in the early stages – work is actually performed on it in order to grasp the big picture.
4. The level of **uncertainty** is very high since it is difficult to accurately define detailed requirements for software projects prior to the start of a software development project. For this reason, an adaptive (**Agile**) approach works best to further elaborate the detailed requirements for a project while the project is in progress rather than trying to define detailed requirements upfront.

Quora

Two Dimensions for Software Engineering

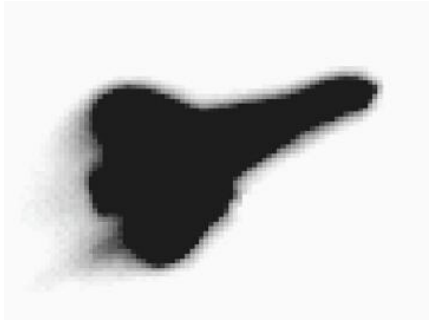
- Essentially therefore, software engineering practices are largely concerned with **managing** relevant processes and with *design* activities, and these can appear in a range of guises – the **management dimension** in contrast to the **technical dimension**.
- Most of the activities involved in software development and evolution tend to use **team-based** processes that embody some form of design element, spanning from an initial choice of a high-level architecture through the choices of test and evaluation strategies.
- Each of these adds yet another layer of **complication**: teams must be organized with regard to aspects such as communication, coordination, and management and design activities are nondeterministic (or "wicked") processes that lead to solutions that are rarely right or wrong [Rittel & Webber 1984; Peters & Tripp 1976].
- Finally, there are also many different **measures of quality** that can be employed when assessing the different choices involved.

https://www.acm.org/education/se2014.pdf

The Human/Management Dimension

- Some researchers argue that software failure statistics are exaggerated.
- Even so, some of these same researchers find a marked difference in such statistics between large and small companies (small companies have better statistics… with fewer people).
- **These studies suggest management (people) problems rather than technical problems are at the heart of the so-called software crisis**.

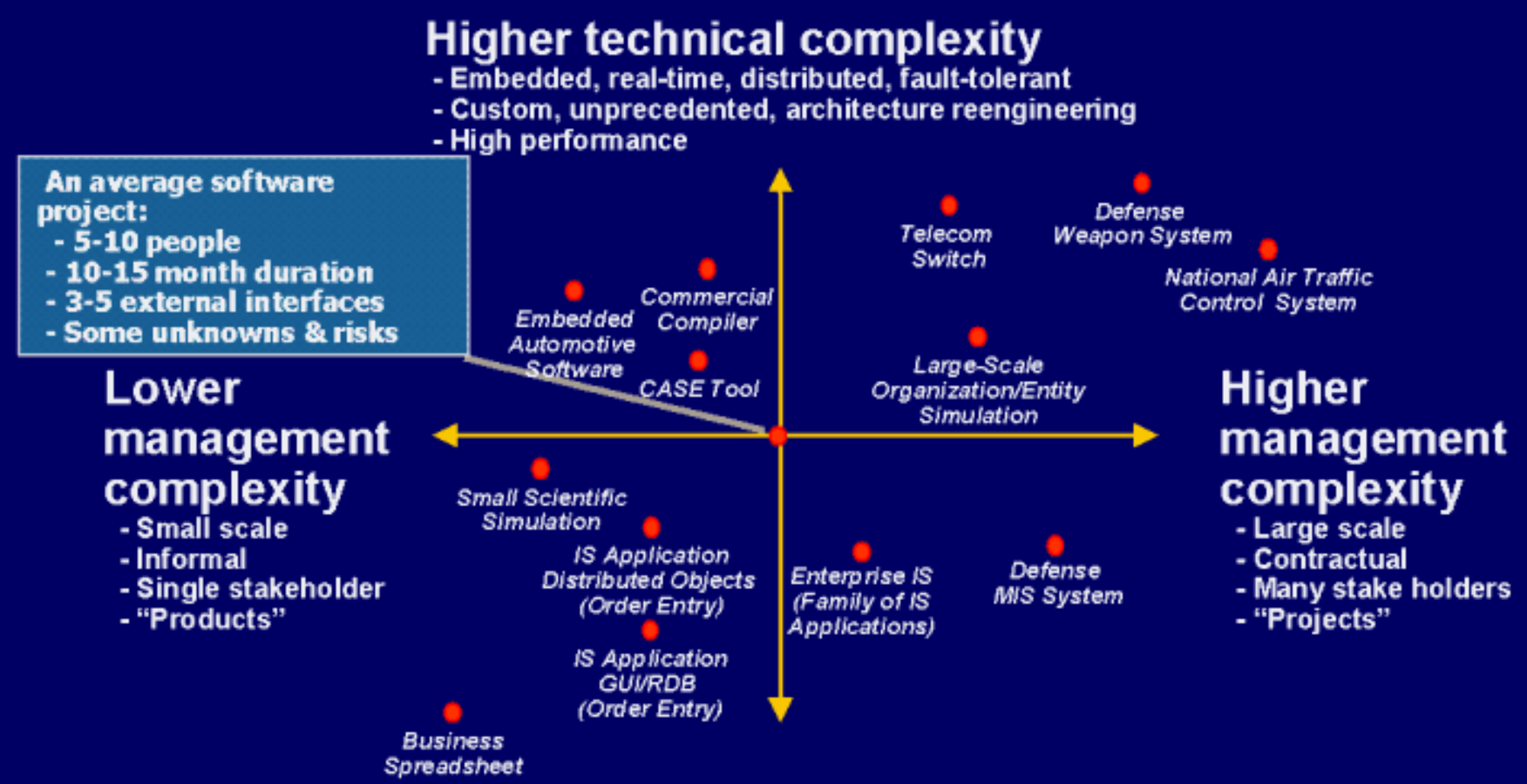
- R. L. Glass, 1996



The Human/Management Dimension

- “The shortage of top-notch programmers threatens to become the limiting factor in software development” (Webster, 01/96)
- “… one of the dirty little secrets of software engineering” [that success in software development depends most upon the quality of the people involved] (Booch, 1981)
- “Excellent developers, like excellent musicians and artists, are born, not made. (Webster, 1996)
  - see Byte Magazine, January, 1996

Dimensions of software complexity



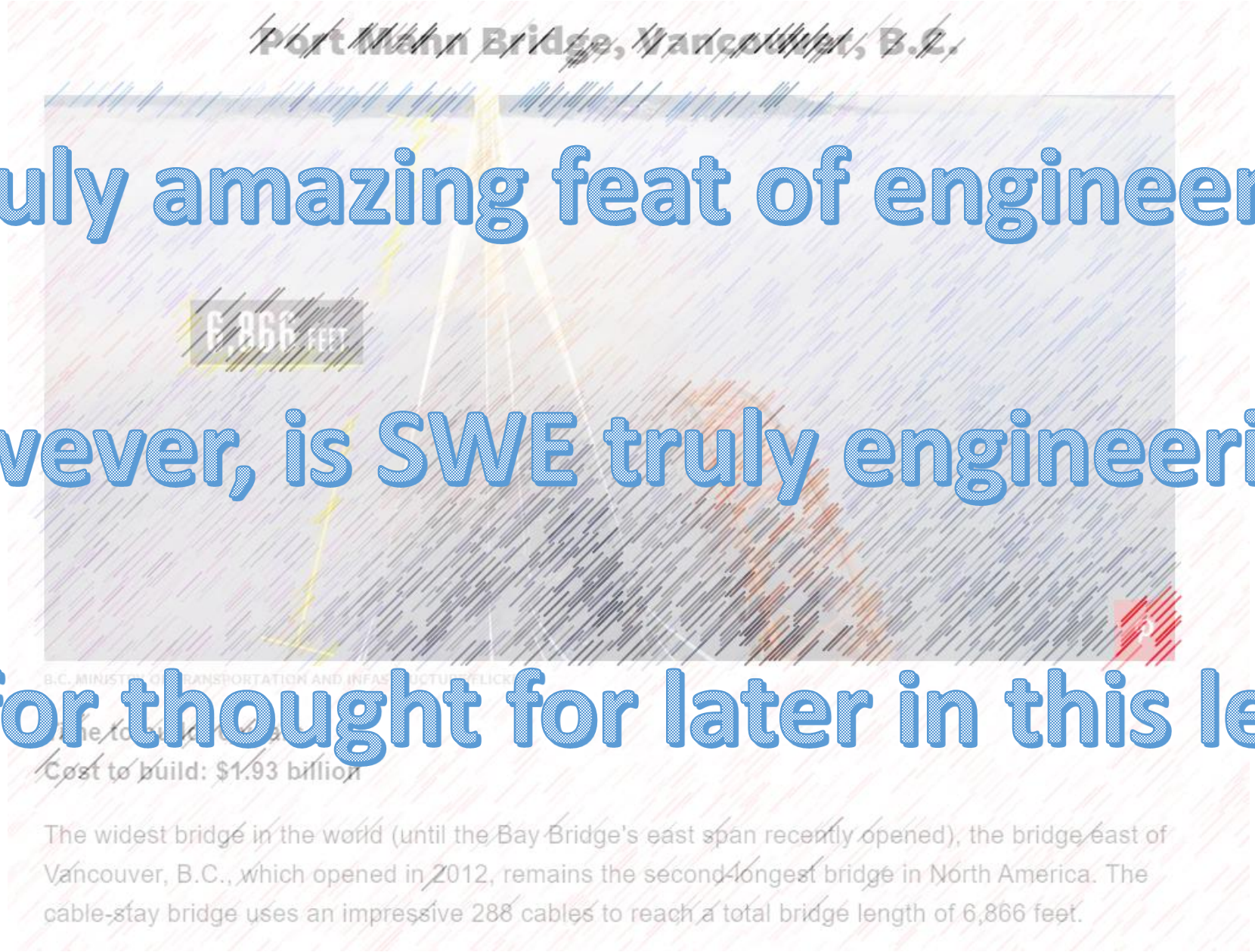
Port Mann Bridge, Vancouver, B.C.

B.C. MINISTRY OF TRANSPORTATION AND INFASTRUCTURE/FLICKR

Time to build: 6 years
Cost to build: $1.93 billion

The widest bridge in the world (until the Bay Bridge's east span recently opened), the bridge east of Vancouver, B.C., which opened in 2012, remains the second-longest bridge in North America. The cable-stay bridge uses an impressive 288 cables to reach a total bridge length of 6,866 feet.

http://www.popularmechanics.com/technology/g2121/the-worlds-25-most-impressive-megaprojects/



A truly amazing feat of engineering.

However, is SWE truly engineering?

Food for thought for later in this lecture.

http://www.popularmechanics.com/technology/g2121/the-worlds-25-most-impressive-megaprojects/

Will your engineering education prepare you for the real world?

1. It <u>does</u> teach you how to actually learn.
2. It <u>does</u> provide you with an initial degree, a jumping-off point.
3. It <u>does</u> teach you to work with others in groups.
4. It <u>does</u> provide technical skills needed for your career
5. It **<u>doesn't</u>** teach you how to be a better communicator.
6. It **<u>doesn't</u>** provide real-world experience.
7. It **<u>doesn't</u>** account for changes in technology.

http://www.electronicproducts.com/Computer_Peripherals/Communication_Peripherals/Will_your_engineering_education_prepare_you_for_the_real_world.aspx?id=201

Position Paper for the Software Engineering Education Workshop at the 13th International Conference on Software Engineering, Austin, Texas, May 13, 1991.

Putting Engineering into Software Engineering Education

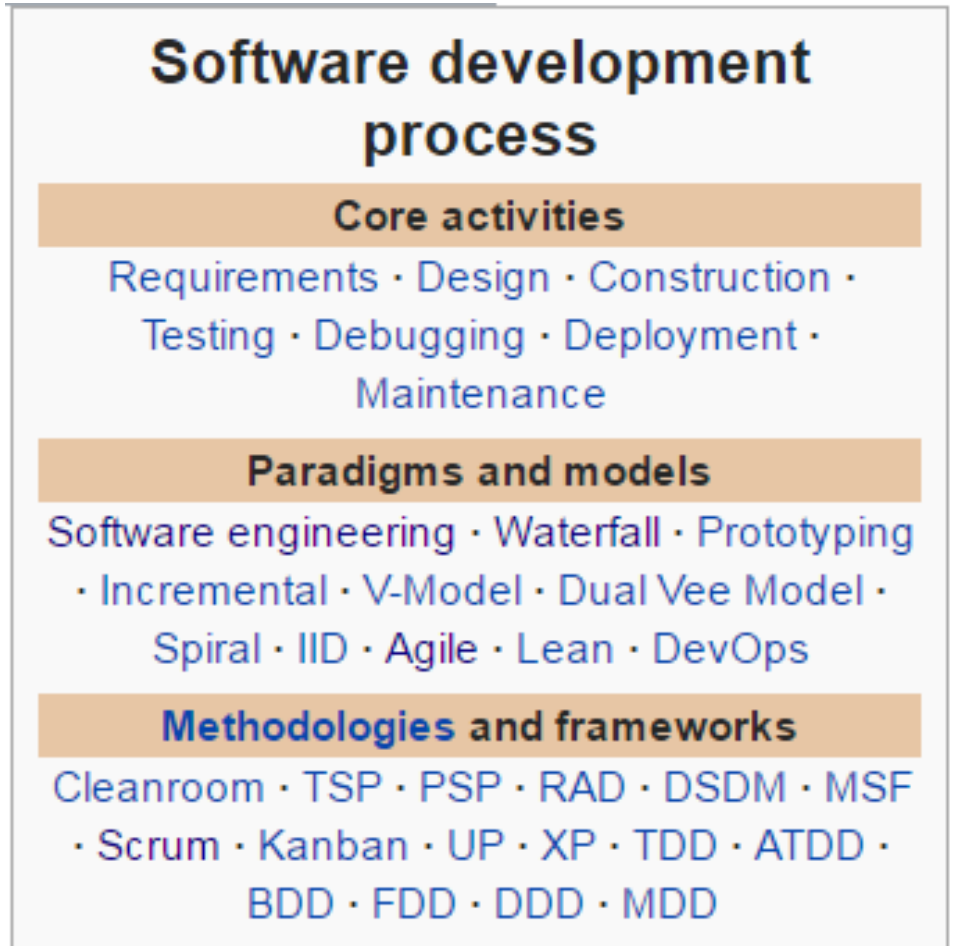
Mary Shaw
Carnegie Mellon University
Pittsburgh, PA

April 1991

Abstract

The current practice of software engineering bears only slight resemblance to the usual standards of engineering practice. This is in part a consequence of the immaturity of the field, but it also results from the failure of software engineering education to instill an engineering mindset in students. This position paper discusses the character of engineering practice with special emphasis on the routine use of reference materials; based on this characterization, it suggests ways software engineering education could better support good engineering practice.

Wikipedia Summary of SWE



Software development process

Core activities

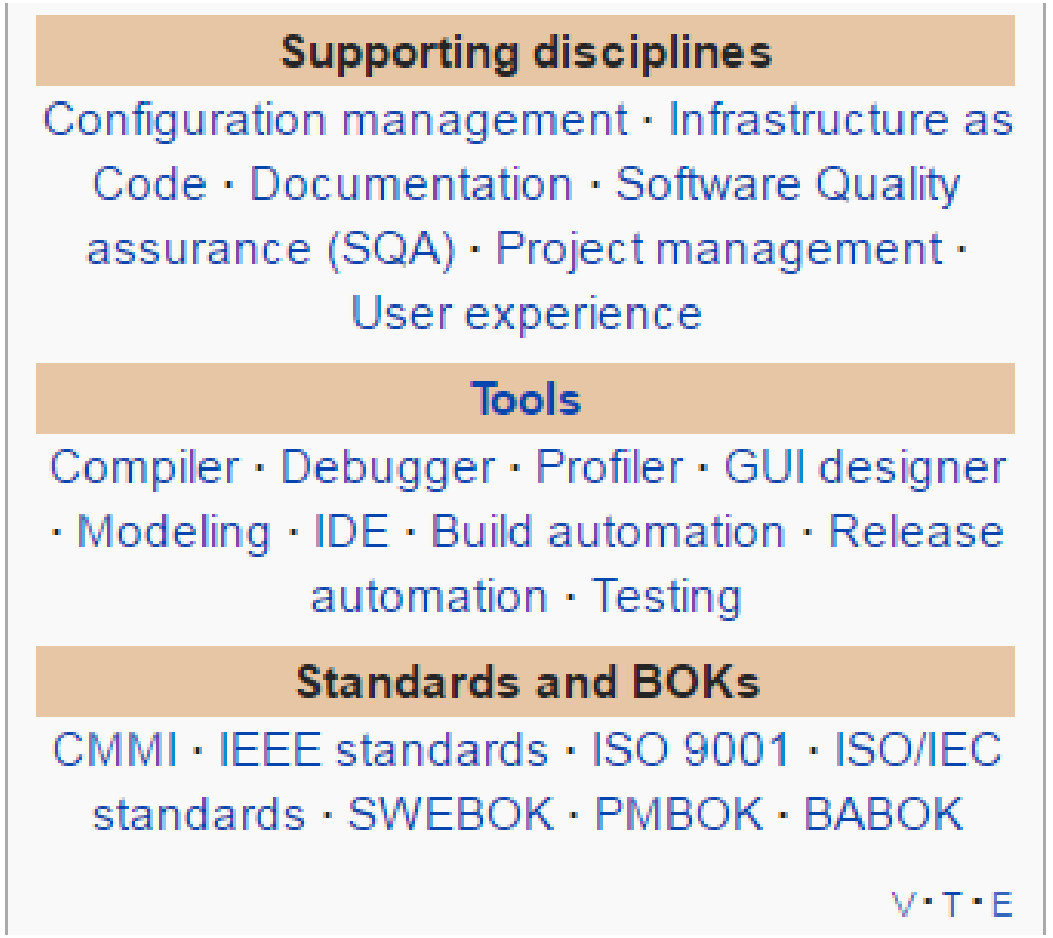
Requirements · Design · Construction · Testing · Debugging · Deployment · Maintenance

Paradigms and models

Software engineering · Waterfall · Prototyping · Incremental · V-Model · Dual Vee Model · Spiral · IID · Agile · Lean · DevOps

Methodologies and frameworks

Cleanroom · TSP · PSP · RAD · DSDM · MSF · Scrum · Kanban · UP · XP · TDD · ATDD · BDD · FDD · DDD · MDD



Supporting disciplines

Configuration management · Infrastructure as Code · Documentation · Software Quality assurance (SQA) · Project management · User experience

Tools

Compiler · Debugger · Profiler · GUI designer · Modeling · IDE · Build automation · Release automation · Testing

Standards and BOKs

CMMI · IEEE standards · ISO 9001 · ISO/IEC standards · SWEBOK · PMBOK · BABOK

V · T · E

READ: https://en.wikipedia.org/wiki/Software_engineering

Week One Assignment and Responsibilities

- Organize yourselves into teams of FIVE and choose a team leader.
- Read the following article from The Atlantic and write a one-page critique with references.
  - Format: Not to exceed one page, single-spaced, 12-point Times New Roman including references.
  - LINK: https://www.theatlantic.com/technology/archive/2015/11/programmers-should-not-call-themselves-engineers/414271/
  - DUE: Next week according to TA submission guidelines.

Presentation Terminated