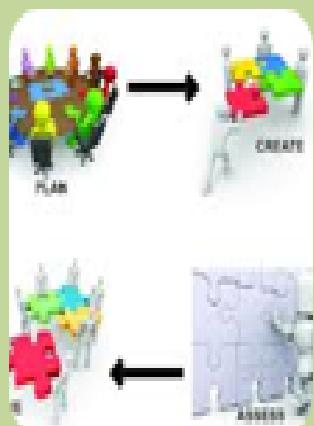


Architecture & Portfolio
Information Management
Security/GRC
Enterprise



Process Management
Project and Resource
Organization and Culture
Team of Teams



**THE
SCALING
PROBLEM**

Operations
Work Management
Product Management
Team



Applications

Infrastructure

IT Value

Founder



Agile IT Management

From Startup to Enterprise (Collaborative Draft)

Table of Contents

Foreword	1
Preface to the Instructor.....	2
The IT industry, and the rise of digital	2
Some cautions.....	5
What this book does not cover.....	6
Some personal history	6
SEIS 660 at the University of St. Thomas	7
Considering a 3rd edition	10
Emergence: vision for a new Agile textbook	11
Assumptions of the reader	13
Acknowledgements	14
Introduction	15
A process of emergence	15
This book's structure	16
Emergence means formalization.....	18
Labs	21
Section I: Founder	23
Chapter 1. IT Value	24
Introduction to Chapter 1	24
What is IT value?.....	25
Defining Information Technology	28
IT services, systems, and applications	32
The IT service lifecycle	37
Defining consumer, customer, and sponsor.....	40
Understanding digital context	42
Conclusion	49
Chapter 2: Infrastructure Management.....	50
Introduction to Chapter 2	50
Infrastructure overview	51
Choosing infrastructure	56
From "physical" compute to Cloud	59
Infrastructure as code	66
Version control: the Agile foundation	69
Policy-based approaches.....	73
Topics in IT infrastructure	73
Conclusion	74
Chapter 3: Application Delivery.....	75
Introduction to Chapter 3	76

Basics of applications and their development.....	77
History of the Agile movement	81
The DevOps challenge	85
Describing system intent.....	88
Test-driven development and refactoring	89
Continuous integration	92
Continuous deployment	94
Conclusion	95
Section I Conclusion	96
Section II: Team	97
Section II introduction	97
Special section: Systems thinking and feedback	99
A brief introduction to feedback.....	100
What does systems thinking have to do with IT?	102
Quality management	105
Chapter 4: Product Management.....	106
Introduction to Chapter 4	106
Why product management?.....	107
Organizing the product team.....	114
Product discovery	120
Assorted topics in Product Management	133
Conclusion	134
Chapter 5: Work Management.....	136
Introduction to Chapter 5	136
Why work management?	138
Introducing Kanban	143
The shared mental model of the work to be done	146
The service desk	150
Lean, Lean product development, and Don Reinertsen	152
From Plan, Build, Run to Demand, Supply, Execute	158
Advanced topics	167
Conclusion	168
Chapter 6: Operations Management.....	169
Introduction to Chapter 6	169
Introducing operations management	170
Monitoring	175
State, configuration, and discovery	177
Capacity and performance management	180
Other operations techniques and tools	181
Advanced topics in operations	181
Conclusion	182

Section II Conclusion	183
Section III: Team of Teams	184
Section III introduction	184
Special section: The elaborated IT organization	185
The perils of "best practices"	185
The IT senior executive	185
Chapter 7: Organization and Culture	186
Introduction to Chapter 7	186
Why organization matters	187
Organizational transition and the Spotify model	188
Organizational theory	195
Product and function	199
Culture.....	206
Topics in organization and culture	212
Conclusion	214
Chapter 8: Project & Resource Management	216
Introduction to Chapter 8	216
Why project management?	219
Principles and theories of project management	228
Agile and project management	238
IT sourcing and contract management	248
IT human resource management	257
IT financial management	264
Topics in IT project management and related fields	277
Conclusion	283
Chapter 9: Process Management	284
Introduction to Chapter 9	285
Why process management?	287
Process basics	292
Process control and continuous improvement	306
Process frameworks.....	312
Lean, Agile, and IT process management	324
Topics in process management	331
Conclusion	337
Section conclusion	338
Section IV: Enterprise	339
Section IV introduction	339
Special section: The IT lifecycles	340
Chapter 10: Governance, Risk, Security, and Compliance	344
Introduction to Chapter 10	344
Governance	346

Enablers	357
Risk management	367
Assurance and audit	374
Security	387
Digital Governance.....	396
Topics.....	410
Conclusion	412
Chapter 11: Enterprise Information Management	414
Introduction to Chapter 11	414
Chapter 11 outline	415
Information and value	416
Data management basics	425
Towards enterprise information management.....	433
Analytics	441
Agile information management	448
Information management topics	455
Conclusion	458
Chapter 12: Architecture and Portfolio	459
Introduction to Chapter 12	460
Why architecture?	462
Architecture practices	478
Architecture domains	501
Agile and architecture	507
Portfolio management.....	518
Architecture standards	521
Topics.....	522
Conclusion	525
Section and book conclusion	526
Appendices & back matter.....	528
The major frameworks	528
Towards a theory of IT management	528
References.....	528

Foreword

Preface to the Instructor

Collaborative

Status: This section is 1st draft since 2015.

Notes: These intro parts may seem rough. As is best practice, the preface and introduction will be given a thorough update at the end of the development process.

More can and should be said about the historic transition to digital; how platform-specific technologies cycle frequently but digital transformation is something much bigger.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

NOTE

This preface is long-winded and represents personal thought processes from some time back. If you are new to this book, it's not a recommended place to start. Will be re-writing and consolidating. (2016-10-03)

The goal of a preface is to position a book, to frame it and help the interested person decide if it makes sense for them. It also helps to tell the story of the book.

This preface covers:

- Some comments on the IT industry and the rise of Agile
- My personal history that led to this project
- A vision for a new, collaborative Agile text

The IT industry, and the rise of digital

Now agile methodologies—which involve new values, principles, practices, and benefits and are a radical alternative to command-and-control-style management—are spreading across a broad range of industries and functions and even into the C-suite.

— Darrell Rigby et al, Harvard Business Review

Let us start this preface with two industry reports:

In September 2015, Minneapolis-based Target Corporation laid off 275 workers with IT skillsets such as business analysis and project management, while simultaneously hiring workers with newer “Agile” skills. As quoted by a local news site, Target stated:

“As a part of our transition to an Agile technology development and support model, we conducted a comprehensive review of our current structure and capabilities... we are eliminating approximately 275 positions and closing an additional 35 open positions. The majority of the impact was across our technology teams and was primarily focused on areas such business analysis and project management.” [\[KARE2015\]](#)

Consider also the following from Jim Fowler, Chief Information Officer at General Electric:

“When I am in business meetings, I hear people talk about digital as a function or a role. It is not. Digital is a capability that needs to exist in every job. Twenty years ago, we broke ecommerce out into its own organization, and today ecommerce is just a part of the way we work. That’s where digital and IT are headed; IT will be no longer be a distinct function, it will just be the way we work. ...

[W]e’ve moved to a flatter organizational model with “teams of teams” who are focused on outcomes. These are colocated groups of people who own a small, minimal viable product deliverable that they can produce in 90 days. The team focuses on one piece of work that they will own through its complete lifecycle...in [the “back office”] model, the CIO controls infrastructure, the network, storage, and makes the PCs run. The CIOs who choose to play that role will not be relevant for long...” [\[Heller2016\]](#)

Fowler echoes the strategic concerns and directions of leading CIOs around the world. And beneath his words lie fundamentally new approaches to delivering IT and digital value. These approaches carry a number of names, but the term “Agile” is prominent, with a number of variants (DevOps, Lean IT, Lean Product Development, Web-scale IT, Lean UX, and so forth). These trends, decades in the making, cannot be dismissed as “flavor of the month” or a fad.

The successes of Agile are notable. Evidence for Agile software development methods, and a related set of approaches that have coalesced into what we might call the digital ecosystem, continues to accumulate.

The annual State of DevOps report presents extraordinary, order of magnitude claims in terms of IT delivery and stability for organizations with mature Agile and DevOps practices [[Puppet2015](#)]:

- 60x fewer failures
- 168x better Mean Time To Recovery
- 30x more frequent delivery
- 200x shorter lead times

The proven effectiveness of these methods in managing many forms of complexity, including problems not usually seen as “IT” or even “digital,” leads to broader and broader appetite for appropriate instruction and curricula.

The new IT-based digital economy is informed top to bottom by an ecosystem of ideas founded on Agile software development. This is accompanied by rethinking or outright rejection of certain practices, including waterfall software development, lengthy “analysis and design” phases, stage-gated project management, and the use of process improvement frameworks. Many are introduced to such practices via formal education, where they are still often presented uncritically.

Startups; large, established firms; and now mid-size companies are embracing the insights and practices of digital transformation, and local workforce requirements are shifting. We believe that academic systems and curriculum owners need up to date understanding and relevant, mainstream offerings appropriate to these trends.

However, Agile + DevOps does not equal all of MIS...

A primary goal of this book, as a textbook, is to prepare the student for a career in this new digital industry. “Industry” is broadly defined as both those industries that offer digital products per se, as well as industries that rely on digital technology instrumentally for delivering all kinds of products. A central theme of the book is that “IT,” considered as a component, represents an increasing proportion of ALL industrial products (both consumer and business facing). This is known as “digital transformation.”

Because of this, the IT industry, and my understanding of it, have transformed dramatically in the past ten years, and the transformation has accelerated even more in the last five.

However, IT remains under-managed and poorly understood relative to peer functions in the enterprise. It struggles with a reputation for expensive inflexibility and Dilbert-esque dysfunction. The DevOps and Agile movements promise transformation, but are encountering an entrenched legacy of:

- enterprise architecture,
- program and project management,
- business process management,
- IT service management practices, and
- IT governance concerns

Understanding and engaging with the challenges of this legacy are an ongoing theme throughout this introductory text.

Some cautions

However, some of the more radical voices in the Agile movement seem to think that the legacy can be simply swept away. The following cautionary message from Mike Burrows shows that, in terms of core Agile philosophy, this would be ill-advised:

“...some will tell you that when things are this bad, you throw it all away and start again. It’s ironic: The same people who would champion incremental and evolutionary approaches to product development seem only too eager to recommend disruptive and revolutionary changes in people-based systems – in which the outcomes are so much less certain.” [Burrows2014], Kindle Locations 827-829.

IT management at scale within an organization is a complex system. Complex systems do not respond well to dramatic perturbations. They are best changed incrementally, with careful monitoring of the consequences of each small change. (This is part of the systems theory foundation underlying the Agile movement.)

This is why the book, especially in the later chapters, starts with relatively straightforward and uncritical presentations of the following topics:

- Project management
- Process management
- Governance, risk, security, and compliance
- Enterprise information management
- Enterprise architecture and portfolio management

While these practices, and their associated approaches and policies, have caused friction with digital and Agile practitioners, they all have their reasons for existing. The goal of this book is to understand their interaction with the new digital approaches, but in order to do this we must first understand them on their own terms. It does no good to develop a critique based on misconceptions or exaggerations about what (for example) "process management" is all about.

Instead, we try to break these large and sometimes controversial topics down into smaller, more specific topics - "lowest common denominators," perhaps:

- Work
- Ordering of tasks
- Task dependencies
- Estimation vs. commitment
- Repeatability
- Defined vs. empirical process control
- Synchronization and cadence
- Resource demand

- Shared mental models
- Mission objectives vs perimeter defense
- Technical debt
- Risk

and so forth.

Distinguishing value: critical approach to frameworks and "best practices" where other texts seem to blindly accept them

So, how do we undertake digital transformation? For longer term, more ambitious goals, while we cannot plan in advance how any given change initiative will play out, we **can** develop awareness.

The idea of the OODA loop is popular in the Agile community. OODA stands for Orient, Observe, Decide, Act, a set of concepts developed by Air Force colonel John Boyd in analyzing the performance of aviators in the Korean War. The faster an actor can move through this cycle, the more effective they will be.

I propose this book as an initial “OO” of the “OODA” loop for those seeking Agile transformation of their IT capabilities. Its purpose is to provide the student with a well documented set of observations on IT’s current challenges that can orient you for more effective decisions and actions in your journey toward IT excellence.

What this book does not cover

This book does not cover specific technologies in any depth. Many examples are used, but carefully framed to not require previous expertise. This is about broader, longer-lifecycle trends.

Some personal history

I teach a survey course, at the University of St. Thomas in St. Paul, Minnesota, at the graduate level in the department of Software Engineering and Information Systems. This program at St. Thomas is the largest such program in the country and is comprised of a broad mix of students. Some are fresh out of a non-technical four-year liberal arts degree, and some have years of IT experience in businesses of various sizes, including the region’s many Fortune 500 corporations.

As you can imagine, this breadth of experience poses some challenges, but also presents opportunities if one can figure out good ways for the students to teach each other—to be covered in the associated lab materials.

My class is entitled "Modern IT Delivery," and has been intended from its creation to cover the management of IT broadly. It serves as a contextual course for the students in their in-depth studies of programming, databases, security, networking, IT processes, and so forth.

Instructor’s note

As of this writing (February 2015) I am looking forward to teaching my class a fifth time. While I do

not consider myself an expert instructor, I have learned a few things about what works in the classroom. I also think I have a good understanding of where the IT industry is going, and what students need to learn to be effective as industry practitioners. In fact, this book is written partly out of a sense that **IT education in this country is broken**.

This may seem like a presumptuous thing for an adjunct faculty to say, but my day job is as a consultant to senior IT leaders at some of the largest corporations in the world. None of them feel well served by the current IT educational system. [1: I admit my bias is US-centric. There is a European discipline called "informatics," that may be closer to what is needed.] "The students coming out don't even understand what source control is," goes one frequent complaint. And Agile methods, if mentioned at all, are presented in a context- and experience-free manner.

This is a problem, as we are starting to see shakeups in the talent market due to larger enterprises adopting Agile. I am aware of hundreds of experienced IT professionals being laid off in my region, due at least in part to Agile transformations. A new pedagogy is called for. (This is why I have also helped found the [Minnesota Agile Study Group](#), a meeting place for local faculty and professionals to interact on these topics.)

SEIS 660 at the University of St. Thomas

I wrote my first book in 2006, *Architecture and Patterns for IT: Service Management, Resource Planning, and Governance (Making Shoes for the Cobbler's Children)*. This work was based on the application of enterprise architecture techniques to the "business of IT," taking ITIL, COBIT, IT portfolio management, and similar inputs as a statement of requirements and analyzing a logical solution. The book was rewritten and released as a 2nd edition in 2011.

In 2013, I was presenting at the SEI Saturn conference in Minneapolis, MN, on the contents of the book and was approached by Dr. Bhabani Misra, the head of the Graduate Programs in Software at the University of St. Thomas in St. Paul. Dr. Misra asked me to teach an "IT Infrastructure" course (SEIS660), which at the time had a very sparse definition:

This course will cover several topics related to IT infrastructure. The course will cover Information Technology Infrastructure Library (ITIL) which is the most widely adopted approach for IT Service Management. It provides a practical framework for identifying, planning, delivering and supporting IT services to the business.

I readily accepted the opportunity. Adjunct positions, while notoriously ill-compensated, are legitimate faculty positions and afford a number of benefits beyond the course stipend. In particular, in these practitioner-focused Masters' programs, one comes into contact with a wide variety of industry professionals and can gain great insight into current trends. Also, there is an aspect of "giving back." Like many teachers before me, I find the work deeply satisfying.

The first semester of the class was well received enough for me to be invited back. However, there were complaints from the students that it was too "theoretical." I was attempting to teach using an enterprise architecture style, with lots of abstractions, that just were not engaging students effectively.

For example, in the ITIL framework, one learns that "an Incident is different from a Problem."

From the perspective of a student new to IT, that is a meaningless semantic distinction. Absent practical reinforcement, it will not be retained after the class, if they even manage to remember it for the final.

I also had a team project approach that immediately started the students out as the IT leadership team of a large corporation. This generated feedback that the students wanted something more practical; they were not going to be immediately hired as senior executives!

I took this feedback seriously, of course. I especially gave thought to a practical aspect, and so started to develop a lab component. This was and is popular with the students, based on the evaluations I get. I also started to think about different approaches for structuring the class that would make more sense for a survey class with a wide spectrum of experience. The fruits of this are detailed below.

As the class progressed, we changed the course description as follows:

SEIS 660 Modern IT Delivery

This course covers the engineering and operation of IT infrastructure, and related IT management practices in both theory and practice. Students participate in building and operating an end to end “IT supply chain” applying current industrial practices, demonstrating how IT services move from idea through production in a practical industrial setting and are managed and improved over time.

This lab simulation is then used to illuminate key IT management topics such as: Cloud – Virtualization – Infrastructure as code – Web-scale IT – Continuous delivery – Change and incident management – Monitoring and service management – IT process management – IT standards – Continuous improvement for IT.

Students will gain hands-on experience with virtualization, systems administration, DevOps, monitoring, collaboration, and industrial IT processes.

As part of the explorations related to developing a new curriculum, I reached out to other faculty and scholars in this space and developed some basic premises:

Sidebar: Proposed premises of the [Agile Study Group](#)

We offer the following premises, as a set of old vs. new contrasts:

Old view	New view
Lean/Agile is a cultural and generational phenomenon that will pass.	Lean/Agile has sound empirical and theoretical foundations. Rather than being seen as a passing fad, it is better understood as the logical culmination of the great systems and management theorists of the 20th century (Deming, Forrester, et al.)

Theory is relevant for core computer science, but less so for IT industry practice.	There are relevant theories for software product management, work execution, and service operations that we as educators should be using to inform our pedagogy.
Teaching students an assortment of foundational theory and functional skills (computation, programming languages, networking, security, and more applied topics) is sufficient.	Students need to understand the collaborative software-based product lifecycle process as a socio-technical system.
The default organizing model of functionally specialized, maximally utilized IT organizations matrixed through projects and processes is effective and needs little further discussion.	Current IT management guidance overlooks critical insights of queueing theory, resulting in gridlock. Project and process management across functionally specialized centers of excellence is being challenged by product-centric approaches (e.g. Amazon's "Two-pizza team" model).
Product design is not our field.	<p>IT is an increasing component of products in general and this is transforming both the practices of IT and product management. It can be argued that all IT system development is a form of product design and delivery.</p> <p>The "business/IT" boundary continues to erode as part of this. The implications of this for traditional IT organizational design and performance management are radical and poorly addressed in research and pedagogy.</p>
Companies vary too greatly in how they employ IT and software graduates for academic training to cover practical issues.	<p>There is an emerging practical industry consensus around how IT is best delivered. Regions where this is most advanced are reaping economic benefit.</p> <p>Certain widely adopted industry practices (source control, continuous delivery, and computing infrastructure automation in particular) need to be better reflected in current instructional approaches. Teaching them as isolated functional topics is insufficient. They should pervade the IT student experience, just as antiseptic practices pervade medical education.</p>

Agile methods can be understood and taught as a subset of project management.	Lean/Agile IT is expanding upstream (via product management and increasing impact on organizational design) and downstream (via DevOps) from traditional project management and is challenging project management's foundational assumptions.
Industry partnerships are essential.	Filtering Lean/Agile through a project management lens tends to reduce it to development-centric Agile practices such as Scrum, and diminishes the breadth of Agile theory and applicability. In fact, the future of project management is itself in question; significant concerns have been raised about the theoretical foundations and practical utility of project management as traditionally understood.
Technology is moving too fast and we continually struggle with keeping up and staying relevant.	Yes, and we need to have additional experiential approaches, such as shared simulation testbeds and effective hands-on labs, for grounding students in current industrial practices so they are well prepared for their industry practicums.

Like any mental model, these contrasts are open to debate, refinement, and even elimination.

Considering a 3rd edition

For three semesters I assigned my book (*Architecture and Patterns*) as a required text for the class. However, I did not write this as a textbook and its limitations became clearer and clearer. In particular, it had a strongly architectural approach, approaching the IT management problem as a series of [views on a model](#). I do not recommend this as a pedagogical approach for a survey class.

I approached my publisher with the idea of a 3rd edition that would pivot the existing material towards being something more useful in class. They agreed to this and I started the rewrite.

However, by the time I was halfway done with the first draft, I had a completely new book. Material from the previous work simply did not fit.

A number of factors converged at this point:

- My view that the "medium is the message" and this extends to choice of authoring approach, intellectual property, DRM, and publisher

- Contacts with local and international faculty and thought leaders, and a desire to openly collaborate with them on making the book as good as possible
- A desire to freely share at least a rough version of the book, both for marketing purposes and in the interests of giving back to the global IT community
- A desire to be able to rapidly update the book with as little friction as possible
- A practical realization that the book might get more uptake globally if available as free and open source IP
- The fact I had already started to [publish my labs on Github](#), and had in fact had developed a reasonably sophisticated "DevOps in a Box" toolchain (the [Calavera project](#), which has attracted collaborators from the US, Spain, and Israel).

Hence this project.

End instructor's note

Emergence: vision for a new Agile textbook

So, what exactly IS this textbook, anyhow?

- It is the first general, survey-level text on IT management with a specific Agile and Lean IT orientation.
- It both covers, and is written using, Agile, Lean, and continuous delivery techniques.
- It has a unique and innovative narrative structure.
- Because it is written with continuous integration and print on demand techniques, it can be continually updated to reflect current industry trends.

Current Agile texts There are hundreds of books on Agile. In this section I will seek to credit specifically collegiate texts that may also be of interest.

- Ashmore, Sondra and Kristin Runyan, [Introduction to Agile Methods](#)
- Cobb, Charles G. [The Project MANAGER'S GUIDE TO MASTERING AGILE: Principles and Practices for an Adaptive Approach](#)
- Limoncelli, Thomas A., Strata R. Chalup, Christina J. Hogan [The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Volume 2](#)
 - While not an academic text per se, does have user questions and exercises at end of chapters; I currently use as supplementary text and am scoping this one to complement it

Most authors of IT/MIS survey texts struggle in my opinion with structuring their narrative. Many start by discussing (in too much depth) various computing fundamentals and then switch to a laundry list of assorted topics, including business needs for computing, project management, programming, database and network management, IT processes, security, and so forth.

A smaller number of authors may start with the business motivation and then go into the

functional areas, but in either case these narratives are rather fragmented.

Waterfall assumptions are found throughout many of these texts, not just as explicit discussion, but embedded pervasively as a mental model, that IT is "planned, built, and run."

While Agile techniques are certainly mentioned, they are typically grafted onto the former narrative. Mostly, Agile is discussed in the context of project management. Questions of end to end flow, product management, Agile infrastructure & operations, culture and organization, the relationship to Lean, and many other such topics go unaddressed.

As mentioned above, educators have a responsibility to effectively respond to the new realities of Lean product development and the end to end Agile transformation looming for enterprises. This can no longer be dismissed as "flavor of the month" or a fad. It is a fundamental transformation of business and society, ultimately based in Mark Andreessen's observation that "software is eating the world."

Finally, there is almost no recognition that the management of information technology differs greatly depending on the **size and maturity of the organization**. College textbooks tend to assume that students are interacting with computers in the context of large, bureaucratic organizations. A smaller percentage may talk about computers and software as products to be developed and marketed in their own right.

influences: Dunbar, Maslow, entrepreneurial literature

Few if any discuss the challenging questions of scaling IT management, and the state transitions it implies. My central educational hypothesis is that **in embracing the scaling problem we can develop an effective pedagogy that can orient even the greenest student.**

The central advantage of this approach is that it is ideal for the new student. There are no technical prerequisites for a course based on this text, although it can be a lot of work for those completely new to IT. Most people can relate to the iconic image of a small team in a garage starting a company, and having it scale over time. Discussions of advanced IT issues such as process frameworks are presented as part of a logical evolutionary progression and thought experiment, rather than simply dropped on the unsuspecting student.

Here is a conceptual illustration of an IT management progression:



As a structuring mechanism, I have tested this in the classroom with some success. I divide a 14-week semester into four major sections, with each having a distinct "persona" for the students to adopt:

- Founder:** The students are in teams of 2, each in a hypothetical startup.
- Team:** The students are in teams of 5-6, enough for specialization to emerge, but still intensely collaborative. A startup that has started to become viable.
- Team of Teams:** The entire class becomes one large organization, a "team of teams" faced with the hard problems of coordination and communication across space and time barriers.
- Enterprise:** Students are back in groups of 5-6, representing the executive leadership of a large enterprise, concerned with IT portfolios, analytics, and the complex dynamics of running heterogeneous IT at massive scale.

The structure of the book is further documented in the next section, Introduction.

Assumptions of the reader

- This book is written at the advanced undergraduate/graduate student level. It is currently available only in English.
- There is no assumption of deep IT experience, but there is an assumption that the person interacts with computers in some depth. They should for example understand the concept of an "operating system." An A+ certification, for example, would more than adequately prepare

someone for this book.

- A person completely unfamiliar with computing will need to supplement their reading as suggested throughout the text. There is a wealth of free and accurate information on IT fundamentals (e.g. compute, storage, networking, programming, etc) and this book seeks more to curate than replicate.

Acknowledgements

Thanks to Dr. Bhabani Misra for asking me to teach at the University of St. Thomas.

Thanks to Stephen Fralippolippi and Roger K. Williams for being the first Github contributors.

Thanks to Jason Baker for text and technical collaboration.

Thanks to Mark Kennaley for guidance on open vs closed loop thinking.

Thanks to Glen Alleman for guidance on modern project management practices.

Thanks to Jeff Sussna for ongoing inspiration, Twitter feedback, discussion question ideas, and sourced quotes.

Thanks to Nicole Forsgren for links to articles on performance management.

Thanks to Evan Leybourn for detailed commentary on project management/Chapter 8.

Thanks to Chris Little and Jabe Bloom for quote provenance.

Thanks to Lorin Hochstein for references.

systems theory influences on this book: Beer, Ackoff

Introduction

Welcome to Agile IT Management: From Startup to Enterprise.

This is a "survey" text intended for the advanced undergraduate or graduate student interested in the general field of applied IT management. It is grounded in basic computing fundamentals, but does not require any particular technical skills to understand. You do not need to have taken any courses in networking, security, or particular programming languages to read this book.

However, you will be presented with material on such topics, including fragments of programming languages and pseudocode, and need to be willing to invest the time and effort to understand.

A process of emergence

Joseph Campbell popularized the notion of an archetypal journey that recurs in the mythologies and religions of cultures around the world. From Moses and the burning bush to Luke Skywalker meeting Obi wan Kenobi, the journey always begins with a hero who hears a calling to a quest...

The hero's journey is an apt way to think of startups. All new companies and new products begin with an almost mythological vision— a hope of what could be, with a goal few others can see...

Most entrepreneurs feel their journey is unique. Yet what Campbell perceived about the mythological hero's journey is true of startups as well: However dissimilar the stories may be in detail, their outline is always the same.

— Steve Blank, *The Four Steps to Epiphany*

A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working simple system.

— John Gall, *The Systems Bible*

One of the most important and distinguishing features of this book is its "emergence model." In keeping with the entrepreneurial spirit of works like Ries' *The Lean Startup*, the book adopts a progressive, evolutionary approach. Your journey through it reflects a process of emergence. Such

processes are often associated with founding and scaling a startup. There are many helpful books on this topic, such as:

- *Nail it then scale it* by Furr and Ahlstrom
- *Scaling Up* by Harnish
- *Startup CEO* by Blumberg
- *The Lean Startup* by Ries

However, this is not a textbook (or course) on entrepreneurship. It remains IT-centric. **And, the book is also intended to be relevant to students entering directly into large, established enterprises.** In fact, it prepares you for working in all stages of growth because it progresses through these four contexts:

- Individual (founder)
- Team
- Small company (team of teams)
- Enterprise

Whether you are in a startup, or on a journey within a larger, established organization, you will (hopefully) become aware as you progress through a broadening context:

- Other team members
- Customers
- Suppliers
- Sponsors
- Necessary non-IT capabilities (finance, legal, HR, sales, marketing, etc)
- Channel partners
- Senior executives and funders
- Auditors and regulators

Part of maturing in one's career is understanding how all these relationships figure into your own overall system of value delivery. This will be a lifelong journey for the student; the author's intent is to provide some useful tools.

This book's structure

Here is a conceptual illustration of an IT management progression:



Elaborating the above outline into chapters, we have:

I. Founder

1. *IT value*. Why do we need computers? What can they do for us?
2. *IT infrastructure* We want to build something. We have to choose a platform first.
3. *IT applications* Let's start building something of use to someone.

II. Team

4. *Product management* What exactly is it we are building? What is the process of discovering our customer's needs and quickly testing how to meet them? How do we better define the product vision, and the way of working towards it, for a bigger team?
5. *Work management* How do we keep track of what we are doing, and communicate our progress and needs at the simplest level?
6. *Operations management* How do we sustain this surprisingly fragile digital service, in its ongoing delivery of value?

III. Team of Teams

The boundary between the "team" and the "team of teams" is a challenging area, and industry responses remain incomplete and evolving. Hence the notation "THE SCALING PROBLEM" on the diagram.

7. *Organization and culture* We're getting big. How do we deal with this? How are we structured? Why this way and not that? How can we benefit from increasing maturity and specialization, while still maintaining a responsive product? What are the unwritten values and rules in our company?
8. *Project and resource management* Work is becoming larger and more complex. How can we track and execute larger segments of it, and coordinate across multiple teams?
9. *Process management* We have a structure. Work needs to flow across it. Product and project teams increasingly demand predictable, consistent delivery of internal services, and synchronization points such as enterprise releases are increasingly critical signals.

I. Enterprise

10. *Security, and governance, risk, and compliance* We need to cope with external forces (regulators, vendor partners, security adversaries, auditors) increasingly defining our options.
11. *Enterprise information management* We've been concerned with data, information, and knowledge since the earliest days of our journey. But at this scale, we have to formalize our approaches and understandings; without that, we will never capture the full value available with modern analytics and Big Data. Compliance issues are also compelling us to formalize here.
12. *Architecture and portfolio* We need to understand the big picture of interacting lifecycles, reduce technical debt and redundancy, accelerate development through establishing platforms, and obtain better economies of scale. We need to define our investment strategy based on a sound understanding of both business needs and technology limitations.

I. Appendices

13. *A review of IT frameworks and standards*
14. *Architectural depictions*
15. *Towards a theory of IT management*

The intent is that the more complex, "enterprise"-scale concerns at the end of the book are presented as part of a logical progression.

Emergence means formalization

The emergence model seeks to define a likely order in which concerns are **formalized**. Any concern may of course arise at any time: the startup founder certainly is concerned with security!

Formalization means at least one or more of the following:

- Dedicated resources
- Dedicated organization
- Defined policies and processes
- Automated tooling

In the author's experience, for example, startups avoid formalized process and project

management. To the extent the concerns exist, they are *tacit* (understood or implied; suggested; implicit). Certainly, a small startup does not invest in an enterprise-class service desk tool supporting a full array of IT management processes, or a full-blown Project Management Office with its own Vice President and associated portfolio automation. Simple work management, with a manual or automated Kanban board, is likely their choice for work management.

But by the time they are a team of teams, specialization has emerged and more robust processes and tools are required. The danger of course is that the formalization effort may be driven by its own logic, and start to lose track of the all-critical business context.

By careful examining these stages of maturation, and the industry responses to them, it is the author's hope that the student will have effective tools to critically engage with the problem of scaling the digital organization.

Instructor's note

I have spent considerable time thinking (agonizing) about the correct ordering of the chapters within these sections. This is possibly the tenth or twelfth version of the chapter ordering. This is an area where I want critical review, but also have strong opinions.

There is benefit to restricting the chapters to 12, as a typical semester runs 14 weeks and the book then fits quite nicely, with one chapter per class and allowing for an introductory session and final exam. (Trying to modfiy the semester system is out of scope for this project.) Of course, a two-semester series, with 2 weeks per chapter, would also work well; each half of the book is also a logical unit.

The governing thought experiment is, "what would I turn my attention to next as my IT-based concerns scale up?" For example, I think work management (implying rudimentary workflow, e.g. Kanban) correctly comes before formalized project management, and project management comes before fully formalized process management (including frameworks such as CMMI, COBIT and ITIL).

Note that this would be a testable and falsifiable theory, if empirical research were done to inventory and characterize organization scaling patterns. If we found (for example) that a majority of organizations formalize governance, risk, security and compliance practices before formalizing project management, that would indicate that those chapters should be re-ordered.

(In my experience, small/medium businesses may have some formal IT project management capability but security & GRC are still tacit, not formalized. This does not mean security is not a concern, but they have not yet hired a CISO, nor instituted formal controls).

The presence of Product Management at an early stage is intended to provoke. Product management is poorly addressed in most current CSci/MIS/IT education, as well as the reigning industry standards. Yet formalizing it is one of the earliest concerns for a startup, and the imperatives of the product vision drive all that comes after.

Interesting evidence to this effect is seen (as of 2015) at the University of California at Berkeley I-School, which has replaced its Project Management course with [Lean/Agile Product](#)

[Management](#), taught currently by the esteemed Jez Humble.

Also, you may notice that **the chapter titles don't necessarily reflect "Agile" terminology**. This is also deliberate, as students are going into a diverse world of much long-established IT. Furthermore, putting "Agile" as a qualifier on each chapter seems gratuitous (e.g. "Agile Operations Management" instead of just "Operations Management.")

The first word of the book's title is "Agile." That declares the orientation, and the proof will be in the reading. My intent is to involve experienced Agile practitioners in contributing to the sections most relevant to them, and I anticipate a high quality end result that is recognizably supportive of the Agile movement's goals and ambitions.

The book however is not a complete dismissal of older models of IT delivery. Wherever possible, Agile is presented as an evolutionary step relative to what has gone before. The specifics of "what's different" are identified, in the interest of de-mystifying what can be a fraught and quasi-religious topic. In the words of Don Reinertsen, you can have "faith based Agile or science based Agile." This book is strictly interested in the latter. Pointers to relevant theory are included, although this is NOT a theoretical text. That will come later... [3: Effective pedagogy requires theory. I seek assistance in both the emerging theory of IT, and relevant theories of pedagogy. I am an amateur in both.]

This emergence model can also be understood as an individual's progression within a larger enterprise. Even if one starts from Day 1 at a Fortune 100 corporation, I believe the progression of one's understanding still progresses through individual, to team, to "team of teams," to enterprise. Of course, one may cease evolving one's understanding at any of these stages, with corresponding implications for one's career.

Some of you may be familiar with the idea of a Minimum Viable Product, Minimum Marketable Release, or similar. In these terms, it is important to understand that each **section** of the book represents an MVP, but not each chapter. One can't begin to deliver IT value without the components discussed in each of Chapters 1-3. The chapters of each section tend to be interdependent, in other words.

More structure w/in structure: Each chapter flows in a roughly "top-down" fashion:

1. "Business" concerns (value, motivation)
2. "Process" concerns (execution, flow)
3. "System" concerns (information & automation)

End instructor's note

Labs

With three chapters in each section, the book can be covered in one intense semester at a chapter a week, although expanding it to a two-semester treatment would allow for more in-depth coverage and increased lab exposure.

I give great credit to both my students and Dr. Misra for challenging me to add a practical component to the course. This required new thinking on my part. How to demonstrate IT management at scale in a lab setting? I have learned that a hands on component is critical, as IT management discussions can be abstract and meaningless to many students. ("Incidents are different from Problems!")

Ten years ago, the best that would have been possible would be paper case studies, perhaps augmented with spreadsheets. But new options are now available. The power of modern computers (even lightweight laptops) coupled with the widespread availability of open source software, makes it is now possible to expose students to industrial computing in a meaningful, experiential way.

I have found great utility in the use of lightweight virtualization technologies such as Vagrant, Virtualbox and Docker. I recommend this approach wholeheartedly. I am always interested in hearing from other instructors who are working from the same approach.

At this writing I maintain my labs publicly on Github. My syllabus and lab structure is under continual improvement. This is my current aspirational plan.

Lecture	Topic	Business lab	Technical lab	Team size
Course introduction	Structure, approach	None	SSH & workstation setup	Individual

SECTION I: FOUNDER

Chapter 1	IT value	Defining an IT product - review SaaS examples	Linux command line	2
Chapter 2	IT Infrastructure	Reviewing current IaaS offerings	Cloud and infrastructure as code - configuring a Vagrant machine manually & w/script that is checked into Github & modified	2
Chapter 3	Applications		Continuous delivery pipeline	2

SECTION II: TEAM

Chapter 4	Product management	User stories. Fail fast/risk mgmt	Behavior-driven development	6-8
-----------	--------------------	-----------------------------------	-----------------------------	-----

Lecture	Topic	Business lab	Technical lab	Team size
Chapter 5	Work management	Scrum, ticketing and kanban		6-8
Chapter 6	Operations management	Service definition	Monitoring (Calavera + Nagios)	6-8

SECTION III: TEAM OF TEAMS

Chapter 7	Organizational structure	Organizational forms & communication channels (paper exercise?)	Continue Nagios	>11 (full class)
Chapter 8	Project & resource management		Open-source project tool	>11 (full class)
Chapter 9	Process management		Open-source ITSM suite	>11 (full class)

SECTION IV: ENTERPRISE

Chapter 10	Security, governance, risk, and compliance			5
Chapter 11	Enterprise Information Management	Portfolio simulation	Porfolio simulation with R?	5
Chapter 12	Architecture and portfolio	Archi (open source modeling tool)		5

I use a central server in teaching my classes, but even that is not necessary. This class can be taught with a zero computing budget, assuming that each team of students at least has access to a modern laptop and a fast Internet connection. As of this writing, I am using free and open source versions of Chef, Jenkins, iTOP, jUnit, Ant, and other tools (see github for the current approach).

Some may question the inclusion of command-line experience, but without some common technical platform it is hard to provide a meaningful “hands-on” experience in the first half of the course. I structure my class on the assumption that the students are at least willing to learn computing techniques, with no prerequisites beyond that. Not even a programming language is required; the Java currently used as a sample is minimal.

Truly beginning students will have to work at the Linux tutorials, but all they need master is basic command line navigation, and I have found this possible with a diverse student body. The labs for the second half of the course use experiential paper-based classroom exercises, GUI-based software, databases, and office productivity tools.

Section I: Founder

This is the introduction to Section I.

In this section, we explore the fundamentals of information technology delivery.

Scenario

You are working in a startup, alone or with one or two partners. You are always in the same room, easily able to carry on a running conversation about your efforts and progress. You have no time or resources to spend on anything except keeping your new system alive and running.

Chapter 1: IT Value

Chapter 1 introduces you to the fundamental concepts of IT value that serve as a basis for the rest of the course. Why do people want computing (IT) services? What are the general outlines of their structure? How do they come into being? How are they changed over time?

All of this is essential to understand for your scenario; you need to understand what computers can do and how they are generally used, if you are going to create a product based on them.

This chapter also covers the basics of how you'll approach building a product. It's assumed you won't develop an intricate, long-range plan but rather will be experimenting with various ideas and looking for "fast feedback" on their success or failure.

Chapter 2: IT Infrastructure

In this chapter, you have a general idea for a product and are ready to start building it. But not so fast... you need to decide some fundamentals first. How will your new product run? What will you use to build it?

It's not possible to begin construction until you decide on your tools. This chapter will provide you an overview of computing infrastructure including Cloud hosting and various approaches to system configuration.

This chapter also presents an overview of source control, as even your infrastructure depends on it in the new world of "infrastructure as code."

Chapter 3: Application delivery

Finally, you're ready to start building something. While this is not a book on software development or programming languages, it's important to understand some basics and at least see them in action.

This is also the "DevOps" chapter; it's not just about writing code, but about the entire end to end system that gets the code you are writing from your workstation, into collaborative environments, and finally to a state where it can be accessed by end users. From source repository to build manager to package repository to production, we'll cover a basic toolchain that will help you understand modern industrial practices.

This section's lab approach

While this is not a book about any particular computing language or platform, we need to describe some technical fundamentals. We'll do so in as neutral a manner as possible. However, this books' accompanying labs are based on [Ubuntu Linux](#) and [git](#), the distributed version control system created by Linus Torvalds to facilitate Linux development.

Chapter 1. IT Value

Introduction to Chapter 1

Collaboration notes

I am concerned that the discussion of Carr & related topics is a bit advanced for introductory students. Or perhaps the flow is off. Thoughts appreciated.

Need to add discussion of Minimum Viable Product - essential precursor to Chapter 2

-ctb

As noted at the outset, you are a small core of a startup. You are building a product of which IT is a significant part (otherwise why are you reading this book/in this class?) Your motivations are entrepreneurial; you want to create a successful business.

You might be housed within a larger enterprise, but the thought experiment here is that you have substantial autonomy to order your efforts.

You want to do something that has a unique IT component. Regardless of your business, you will need accounting and legal services at a minimum, and very quickly payroll and HR, and so forth. Those things can (and should) be purchased as commodity services, if you are a small entrepreneur (I am not aware of any convincing arguments to the contrary, unless you are absolutely on the smallest of shoestring budgets and can work 100 hour weeks).

Your unique value proposition will be expressed to some degree in unique IT software. While this software may be based on well understood products, the configuration and logic you construct will be all your own. Because of this, you are now a producer (or soon to be) of IT services.

Before we can talk about building and managing information technology (IT), we need to understand what it is and why people want it. We'll start this chapter by looking at an IT value experience that may seem very familiar. Then we'll dig further into concepts like the "IT stack" and the "IT service" and how they change over time.

Chapter outline:

- An IT value experience
- What is "information technology"?
- The IT "service" and the IT "stack"
- The IT service
- IT changing over time
- Conclusion

Learning objectives for this chapter:

- Explain "IT value" in everyday terms
- Distinguish between IT service and IT system
- Discuss how IT services change over time

What is IT value?

An IT value scenario



Dinner out tonight? [4: Image credit <https://www.flickr.com/photos/garryknight/700317885/>, downloaded 2016-09-14, commercial use permitted]

Consider the following scenario:

A woman is wondering if she can afford to dine out that evening.

- She uses her mobile device to access her banking information and determines that in fact she does have enough money to do so.
- She also uses her mobile device to make a reservation and contact some friends to join her.
- Finally, she uses social navigation software to avoid heavy traffic, arriving at the restaurant in time for an enjoyable evening with her friends.

Information technology pervaded this experience. The origins, layers and complex connections of the distributed systems involved are awe-inspiring to consider.

IMPORTANT Don't worry about the technological terms for now

This is an introductory text. You may see terms below that are unfamiliar (Model-View-Controller, IP, packet switching). If you are reading this online, you can follow the links, but it's not required.

As you progress in your career, you will always be encountering new terminology. Part of what you need to learn is when it's important to dig into it, and when you can let it pass for a time.

You should be able to understand the gist presented below that these are complex systems based on a wide variety of technologies, some of them old, some new.

The screen on her cell phone represents information accessed and presented via a [Model-View-Controller framework](#), implemented in the latest version of [Javascript](#), running on an [interpreter](#) that would have taxed a [mainframe](#) thirty years ago. The communication with her bank's central systems is supported by [4G LTE](#) data which in turn relies on the high-volume [IP backbone](#) networks operated by the [telecommunications carriers](#), based on research into [packet switching](#) now approaching 50 years old.

The application operating on the cell phone interacts with core banking systems via sophisticated and highly secure [middleware](#), crossing multiple [network](#) control points. This middleware talks in turn to the customer demand deposit system that still runs on the mainframe.

The mainframe is now running the latest version of [IBM's zOS operating system](#) (a direct descendant of [OS/360](#), one of the most significant operating systems in the [history of computing](#)). The "customer demand deposit" banking application running on the mainframe is still based on code written in the lowest level [assembler](#). Some of the comments in this code date back to the 1970s. It has been tuned and optimized over the decades into a system of remarkable speed and efficiency. Although replatforming it is periodically discussed, the cost/benefit ratio for such a project has to date not been favorable.

The reservation system looks similar on the mobile device, but the network routes it to a large [Cloud](#) data center hosting the reservation system. The back end application here is very different from the banking system; the [programming languages](#) are newer, the [database](#) is structured very differently, and the operating system is [Linux](#).

Finally, the navigation software looks much like the reservation system, as it too is based on the Cloud. However, the system is much more active, as it is continually processing inputs from millions of drivers in thousands of cities, and updating traffic maps for those drivers in real time so that they can choose the most optimal route to their destinations (e.g., dinner). The capabilities of this system are comparable to an air traffic control system, and yet it is available as a free download for our IT user.

The resulting value is clear:

- In an earlier era, our user might have stayed in, for fear of bouncing a check, or might have gone out and dined beyond her means.
- The phone line at the restaurant might have been busy, so she might have risked showing up with no reservation.
- Before texting and social media, she might not have been able to reach her friends as easily.
- Without the traffic application she might have run into a huge midtown traffic jam and been half an hour late.

Clearly, information technology added value to her life and helped maximize her experiences of social enjoyment.



Digital made this gathering easier [5: Image credit <https://pixabay.com/en/friends-celebration-dinner-table-581753/>, downloaded 2016-09-14, commercial use permitted]

Various forms of IT value

SORs, SOEs, SOIs

Transactional

Social

The startup problem

most startups can only guess who their customers are and what markets they are in.

Blank, Steve (2013-10-02). The Four Steps to the Epiphany (p. 37). K&S Ranch. Kindle Edition.

some basics on the lean startup approach

we have to have a product dev pipeline to even **start** testing and learning "you're going to start building your product even before you know whether you have any customers for it."

Blank, Steve (2013-10-02). The Four Steps to the Epiphany (p. 44). K&S Ranch. Kindle Edition.

then we dig further into the problem of product and customer discovery in chapter 4

Defining Information Technology

What is IT, anyways?

We've started this book in the previous section, by first defining IT value. This is deliberate. But what is IT, anyways?

- The computers? The networks?
- The people who run them?
- That organization that loves to say "no" and is always slow and expensive?

None of these are how this book defines "IT." Although this is not a technical book on computer science or software engineering, the intent is that it reflects and is compatible with academic theory.

Therefore, "information technology" is defined as the edifice of theory and practice based on the work of [Claude Shannon](#), [Alan Turing](#), [Alonzo Church](#), [John von Neumann](#), and the other pioneers who defined the central problems of [information theory](#), [digital logic](#), [computability](#), and [computer architecture](#).

Discussions of "information technology" become contentious because some think of the traditional organization, while others think of the general problem area. IT has a long history as a corporate function, a single hierarchy under a powerful Chief Information Officer. This model has had its dysfunctions, including a longstanding reputation for being slow and expensive. Often, when one encounters the term "IT," the author using the term is referring to this organizational tradition.

We are less interested in the future of IT as a distinct organizational structure. There are

many different models, from fully centralized to fully embedded. Organizational structure will be discussed in Chapter 7.

For this book, we define “Information technology” in terms of its historic origins. We look to IT’s common origins in automating the laborious and error prone processes of computation, through the application of digital logic technologies based on information transmission.

Regardless of organizational form or delivery methods, IT is defined by these origins. And there are notable common threads throughout this problem domain: the fragility and complexity of these systems, the need for layered abstractions in their management, and more.

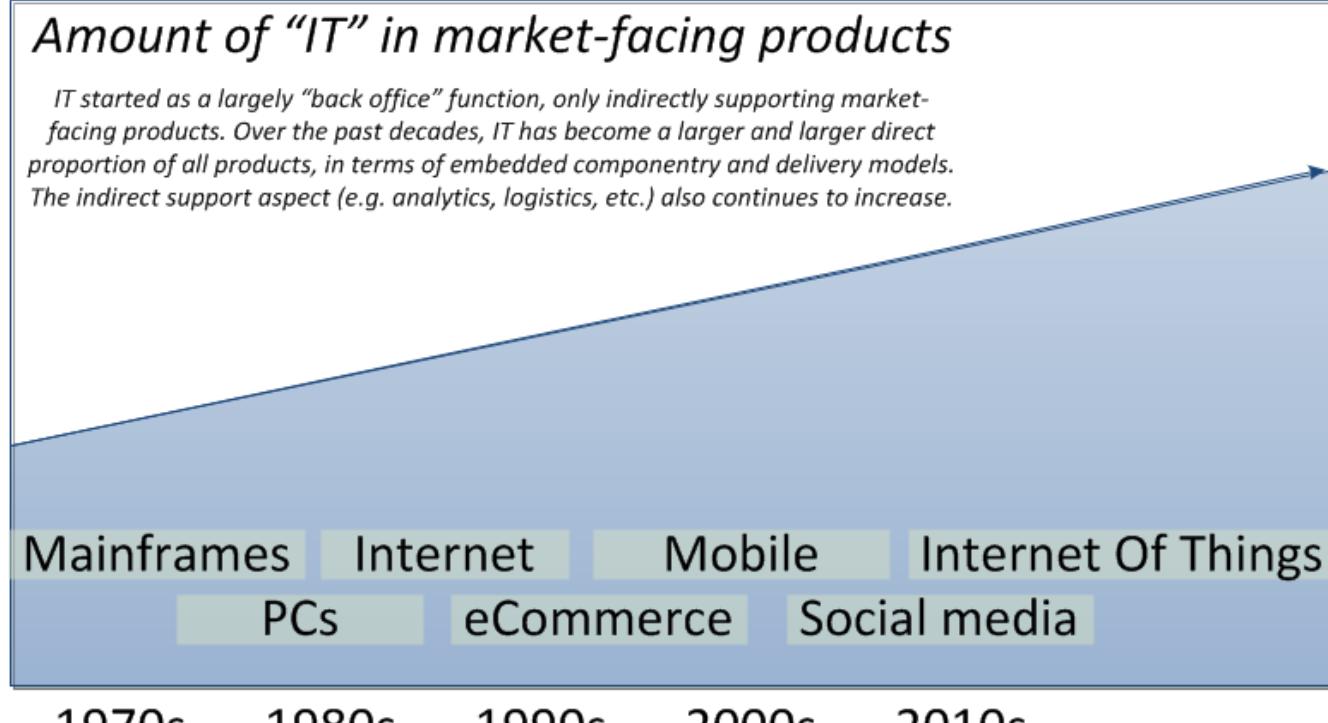
It does not matter if the application developers ultimately report up through the CIO, the CMO, the CFO, or the COO. Their daily experience remains largely the same. The dynamics of their management remain the same. Executives who seek to take control of IT so they can “remove that old bureaucracy” are well advised to be cautious; the bureaucracy emerged for a reason. More on this in subsequent chapters.

IT and digital transformation

The digital realm is infusing the physical realm, like tea in hot water.

— Jeff Sussna, Designing Delivery

IT increasingly permeates business operations and social interactions. The breadth and depth of IT support for virtually all domains of society continues to expand. Lately, this is known as digital transformation.



As IT increases its pervasiveness, it becomes difficult to detangle and analyze it. However, doing so

remains essential because without a line of sight back to its intellectual foundations, any technology risks becoming magic, and its users risk becoming a [cargo cult](#).

Sidebar: Cargo cult

During World War II, South Pacific native peoples had been exposed abruptly to modern technological society with the Japanese and US occupations of their islands. Occupying forces would often provide food, tobacco, and luxuries to the natives to ease relations. After the war, various tribes were observed creating simulated airports and airplanes, and engaging in various rituals that superficially looked like air traffic signaling and other operations associated with a military air base.

On further investigation, it became clear that the natives were seeking more “cargo” and had developed a magical understanding of how goods would be delivered. By imitating the form of what they had seen, they hoped to recreate it.

In 1974, the noted physicist Richard Feynman gave a speech at Caltech in which he coined the immortal phrase “cargo cult science.” His intent was to caution against activities which appear to follow the external form of science, but lack the essential understanding at its core. Similar analogies are seen in business and IT management, as organizations adopt tools and techniques because they have seen others do so, without having fundamental clarity about the problems they are trying to solve and how a given technique might specifically help.

It should be noted, as with most archetypes of this sort, that there are questions around the accuracy of the original accounts and Western interpretations and mythmaking around what was seen. However there is no question that the phenomena were observed, and that “cargo cult thinking” is a useful cautionary metaphor. We will return to it periodically throughout this book.

Nicholas Carr, in his controversial *Harvard Business Review* article "IT Doesn't Matter," recognized that IT was becoming commoditized in an important sense. As Cloud providers started to offer utility-style computing, the choice of particular vendors of computers was no longer strategic. Looking to history, Carr argued that just as businesses no longer have “Vice Presidents for Electricity,” so businesses no longer need Chief Information Officers or dedicated IT departments.

Carr has insight—there is no question IT is becoming pervasive—but he ultimately reflects a narrow view of what “IT” is. If “IT” were merely computation at the lowest level—just shuffling bits of information around, doing a little math—then perhaps it could be embedded throughout a business like electricity.

But IT has emergent aspects that are not comparable to electrical power. As it pervades all dimensions of business operations, it brings its concerns with it: complexity, fragility, and the skills required to cope with them.

One watt of electrical power is like any other watt of electrical power, and can usefully be seen as a commodity. We can use it to run toasters, hair dryers, or industrial paint mixers, and there is little

concern (beyond supply and demand management) that the consumption of power by the paint mixer will affect the toaster. It's also true that one cycle of computing, in a certain sense, is like any other cycle. But information technology systems interact with each other in surprising and unpredictable ways, orders of magnitude more complex than electrical power grids. (This is not to imply the modern electrical grid is a simple system!)

IT also radically transforms industries: from retail to transportation to manufacturing to genetics. Applied software-centric IT is unleashing remarkable economic disruption.

A lawyer may depend on a cell phone, and (in keeping with Carr) beyond its provision as a commodity service, needs little else to deliver the legal strategies a firm needs. A graphic designer may use computerized graphic tools, but these have become relatively standardized and commoditized in the past twenty years, and probably are not a source of competitive advantage in the quest for new marketing clients.

On the other hand, consider a text analytic algorithm that replaces thousands of paralegals, resulting in order-of-magnitude more accurate legal research in a fraction of cost and time. This is strategic and disruptive to the legal community. A superior supply chain algorithm, and the ability to improve it on an ongoing basis, may indeed elevate a logistics firm's performance above competitors. In cases like these—and they seem to be increasing—IT matters very much.

In the digitally transforming economy, traditional “back office” IT organizations find themselves called on to envision, develop, and support market-facing applications of IT. And what starts with one market-facing use case can quickly expand into entire portfolios. It is such cases that are of particular concern in this book.

Ultimately, it is possible that IT is the **most** strategic capability an organization can invest in. As the editor in chief of *IEEE Software* notes [\[Spinellis2015\]](#),

other industries are also producing what's in effect software (executable knowledge) but not treating it as such . . . Although many industries have developed their own highly effective processes over the years, software engineering maintains an essential advantage. It has developed methods and tools that let even small teams manage extremely high complexity . . . This advantage is important because the complexity in non-software activities is also increasing inexorably . . . [T]he time has come to transform our world... by giving back to science and technology the knowledge software engineering has produced.

— Diomidis Spinellis, IEEE Software

This ability to manage complexity, to turn tacit into explicit and formalize the previously unstructured, is an essential aspect of digital transformation.

Are you "doing IT?"

So, how do we define an IT problem, as opposed to other kinds of business problems? An IT problem is any problem where you are primarily constrained by your capability and

understanding of IT.

- If you need computer scientists or engineers who understand the fundamentals of information theory and computer science, you are doing IT.
- If you need people who understand when your information-centric problems might need to be referred to such theorists and engineers, you are likely doing IT.
- If you need people who are skilled in building upon those fundamentals, and operating technical platforms derived from them (such as programming languages, general purpose computers, and network routers), you are doing IT.

Regardless of whether IT is housed under a traditional CIO, an operations capability, a Chief Marketing Officer, or a “line of business”, when it is critical to operations certain concerns inevitably follow:

- Requirements (i.e. your intent for IT)
- Sourcing and provisioning
- IT-centric product design and construction
- Configuration and change management
- Support
- Improvement

Executives who take control of information technology in hopes of making it more "agile" are often surprised to find that these concerns were not mere bureaucracy, but instead had well grounded origins in past failures. Ignoring these lessons is perilous.

And yet, the traditional, process-heavy IT organization does seem dysfunctional from a business point of view: a central theme of this book.

IT services, systems, and applications

Inside an IT service

Let's examine our diner's value experience in more detail, without getting unnecessarily technical, and clarify some definitions along the way.

The first idea we need to cover is the "moment of truth." In terms of information technology, this English-language cliché represents the user's experience of value.

IMPORTANT

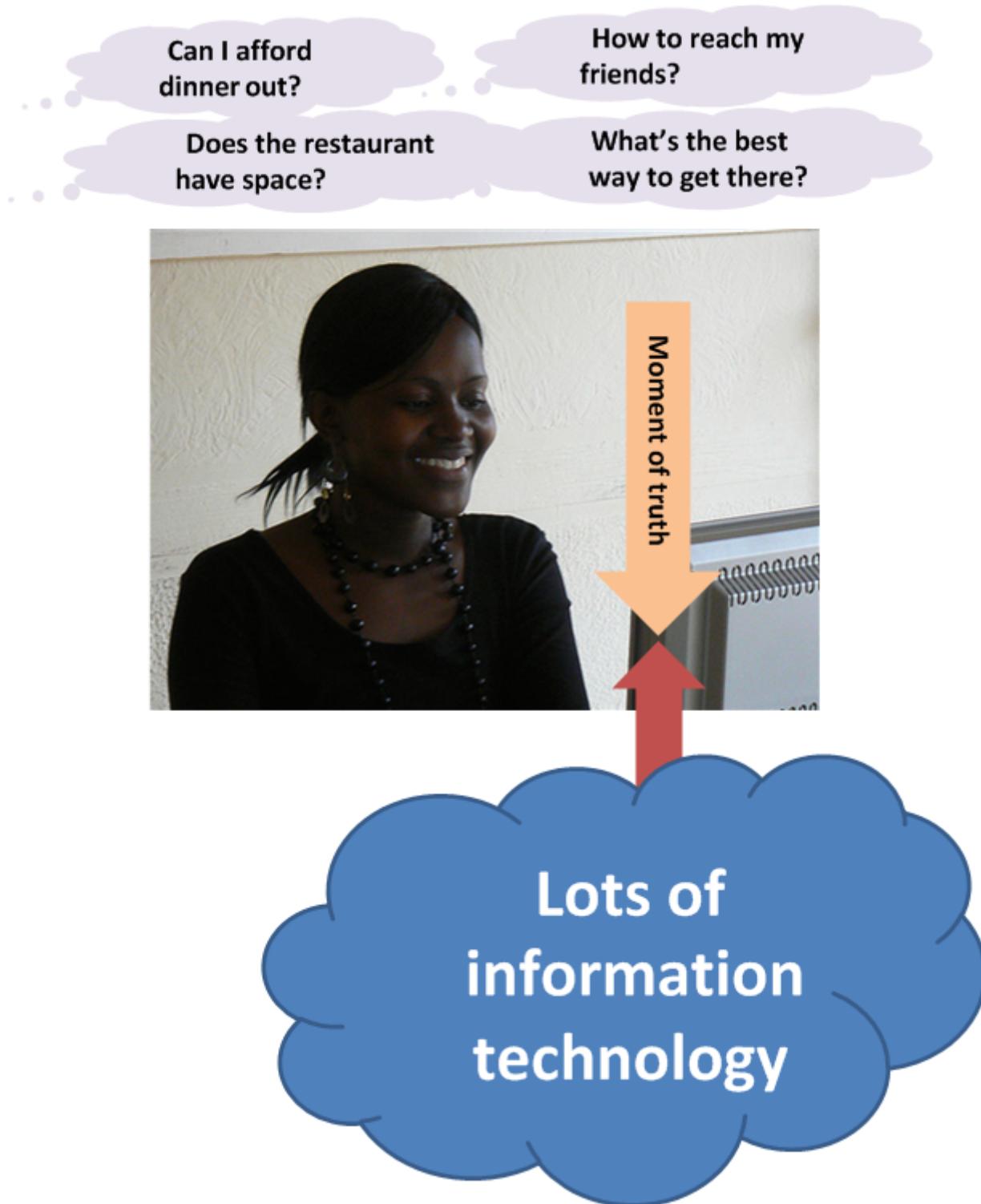
The "moment of truth" represents the user's experience of value, from a product, good, or service.

In the example, our friend seeking a relaxing night out had several moments of truth:

- Consulting her bank balance, and subsequent financial transactions also reflecting what was stated to her

- Making a reservation and having it honored on arrival at the restaurant
- Arriving on time to the restaurant, courtesy of the traffic application

Each of these individual value experiences was co-created by our friend's desire for value, and the response of a set of IT resources.



In order to view her balance, our user is probably using an application downloaded from a "store" of applications made available to her device. On her device, this "app" is part of an intricate set of components performing functions such as:

- connecting to the phone network

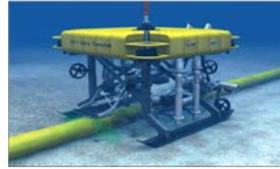
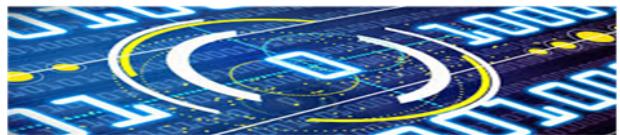
- securely connecting over the phone network to the Internet and then to the bank
- identifying the user to the bank's systems
- requesting the necessary information (in this case, an account balance)
- receiving that information and converting it to a form that can be represented on a screen
- finally, displaying the information on the screen

The application, or "app," downloaded to the phone plays a primary role, but is enabled by:

- the phone's operating system and associated services
- the phone's hardware
- the telecommunications infrastructure (cell phone towers, long distance fiber optic cables, switching offices, and much more)

Of course, without the banking systems on the other end, there is no bank balance to transmit. These systems are similar, but on a much larger scale than our friend's device:

- Internet and middleware services to receive the request from the international network
- Application services to validate the user's identity and route the request to the appropriate handling service
- Data services to store the user's banking information (account identity and transactions) along with millions of other customers
- Many additional services to detect fraud and security attacks, report on utilization, identify any errors in the systems, and much more.
- Physical data centers full of computers and associated hardware including massive power and cooling infrastructure, and protected by security systems and personnel.



Consider: what does all this mean to our user? Does she care about cell phone towers, or middleware, or triply-redundant industrial-strength Power Distribution Units? Usually, not in the least.

Therefore, as we study this world, we need to maintain awareness of her perspective. Our friend is seeking some value that IT uniquely can enable, but does not want to consider all the complexity that goes into it. She just wants to go out with friends. The moment of truth depends on the service;

the service may contain great complexity, but part of its success lies in shielding the user from that complexity.



IMPORTANT

Always remember the user's experience. Information technology has a well deserved reputation for being too complicated for end users—for example, trying to do something that should be simple, and finding oneself in a technical conversation about network settings.

What vs. how

This fundamental tension between **what** a system is supposed to do, versus **how** it does it, pervades IT management and will likely define your career. "Don't trouble me with the details, just give me the results" is the overall theme, and we encounter this reaction to complexity in many aspects of life. You may perceive similar concerns in:

- demand vs. supply
- problem vs. solution
- outside-in vs. inside-out thinking
- service vs. system
- black-box vs glass box

Terminology is important. We need to have a more precise way of describing the information technology, beyond just saying there is "lots" of it. A variety of terms are used in this text:

- IT service
- Application
- IT system
- IT infrastructure

We also see discussion of components, resources, subsystems, assets, and many more terms.

WARNING

There are many debates around these definitions. Sometimes these debates are helpful in clarifying the terminology you want to use on your team. But sometimes the debates don't add any value. Beware of anyone who claims there is a "best practice" here.

In general, in this book, we will use the following definitions:

- An IT service is defined primarily in terms of WHAT not HOW
- Defining an IT system may include a discussion of both WHAT it does and HOW it does it
- An "application" usually means some IT service or system for end users who are not primarily concerned with IT other than wanting to get something done with it (e.g. go out to dinner)
- "Infrastructure" usually means some IT service or system that primarily supports OTHER IT services or systems (e.g. a network "service" is not usually useful to end users without additional application services.)

Finally, the concept of the "IT stack" is important. Notice how the different technology layers appear "stacked." Layered approaches to understanding IT are common; see Further Reading for useful references.

Sidebar: The Digital Services Playbook

The IT service lifecycle

We've established that the IT service is based on a complex stack of technology, from local devices to global networks to massive data centers. Software and hardware are layered together in endlessly inventive ways to solve problems people did not even know they had ten years ago.

But these IT service systems must come from somewhere. They must be designed, built, and operated, and (as any smartphone user can testify) they are continually improved over time.

A simple representation of the IT service lifecycle is shown in the figure below.

1. It starts with an idea. Someone has an insight into an IT-enabled value proposition that can make a profit, or better fulfill a mission.
2. The idea must garner support and resources so that it can be built.
3. The idea is then constructed, at least as an initial proof of concept or Minimum Viable Product (in the language of Ries' *Lean Startup*.) Construction is assumed to include an element of design; in this textbook, **construction and design are not represented as two separate phases**.
4. There is a critical state transition however that will always exist. Initially, it is the change from OFF to ON when the system is first constructed. After the system is ON, there are still distinct changes in state when new features are deployed, or incorrect behaviors ("bugs" or "defects") are rectified.
5. The system may be ON, but it is not delivering value until the user can access it. Sometimes, that may be as simple as providing someone with a network address, but usually there is some initial "provisioning" of system access to the user, who needs to identify themselves.
6. The system can then deliver services (moments of truth) to the end users. It may deliver millions or billions of such experiences, depending on its scale and how one might choose to count the subjective concept of value experience.
7. The user may have access, but may still not receive value, if they do not understand the system well enough to use it. Whether via a formal service desk, or informal social media channels, users of IT services will require and seek support on how to maximize the value they are receiving from the system.
8. Sometimes, support requests indicate that something is wrong with the system itself. If the system is no longer delivering value experiences (bank balances, restaurant reservations, traffic directions) then some action must be taken promptly to restore service.
9. All of the previous states in the lifecycle generate data and insight that lead to further evolution of the system. There is a wide variety of ways systems may evolve: new user functionality, more stable technology, increased system capacity, and more. Such motivations result in new construction and changes to the existing system, and so the cycle begins again.
10. ...Unless... the system's time is at an end. If there is no reason for the system to exist any longer, it should be retired.

NOTE

System retirement is often more complex and expensive than expected, and there are many examples of systems surviving well beyond the point they deliver value.

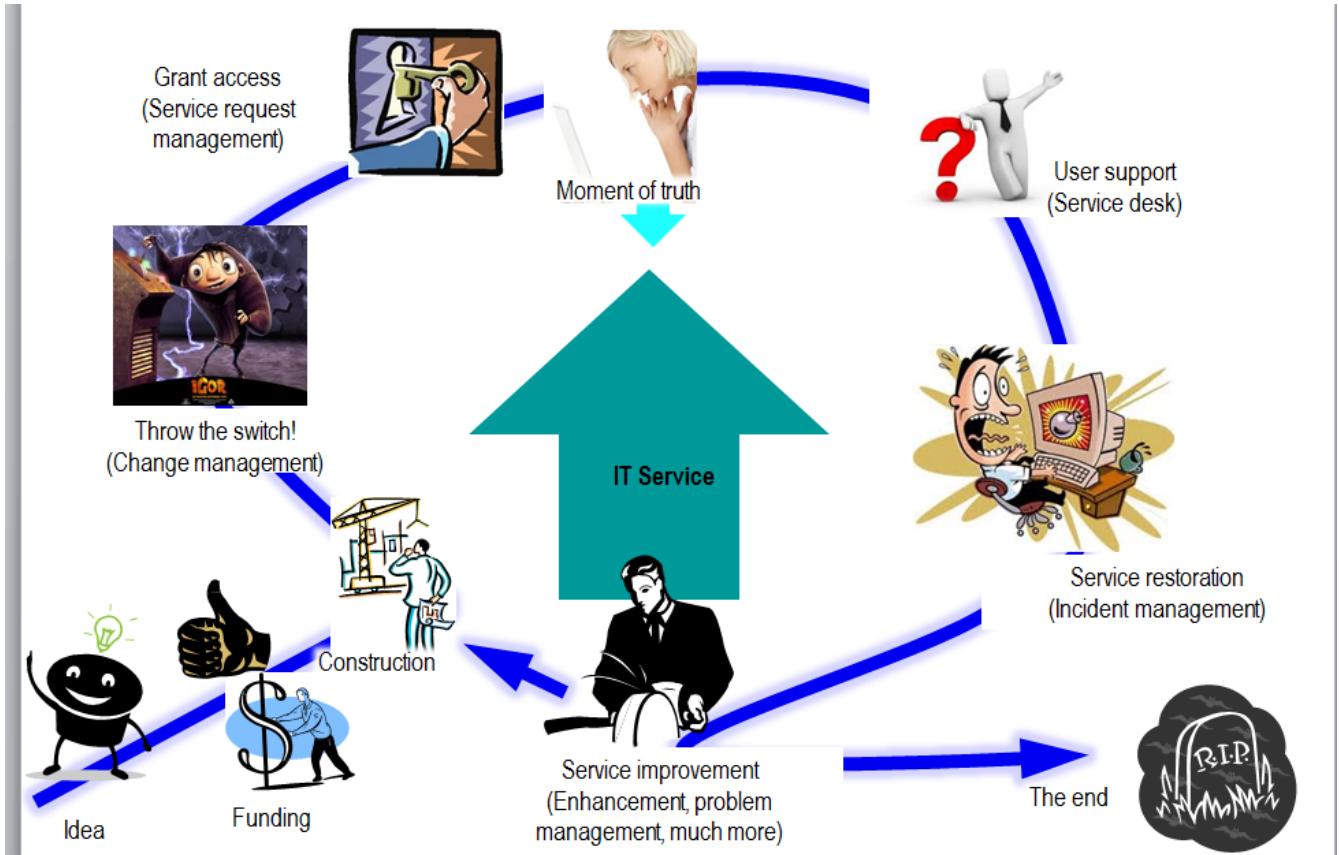


Figure XX - The essential states of the IT service

So we see that the IT service evolves over time, through many repetitions ("iterations") of the improvement cycle. This entire process, from idea to decommissioning ("inspire to retire") can be understood as the *service lifecycle*.

The IT service evolves over time

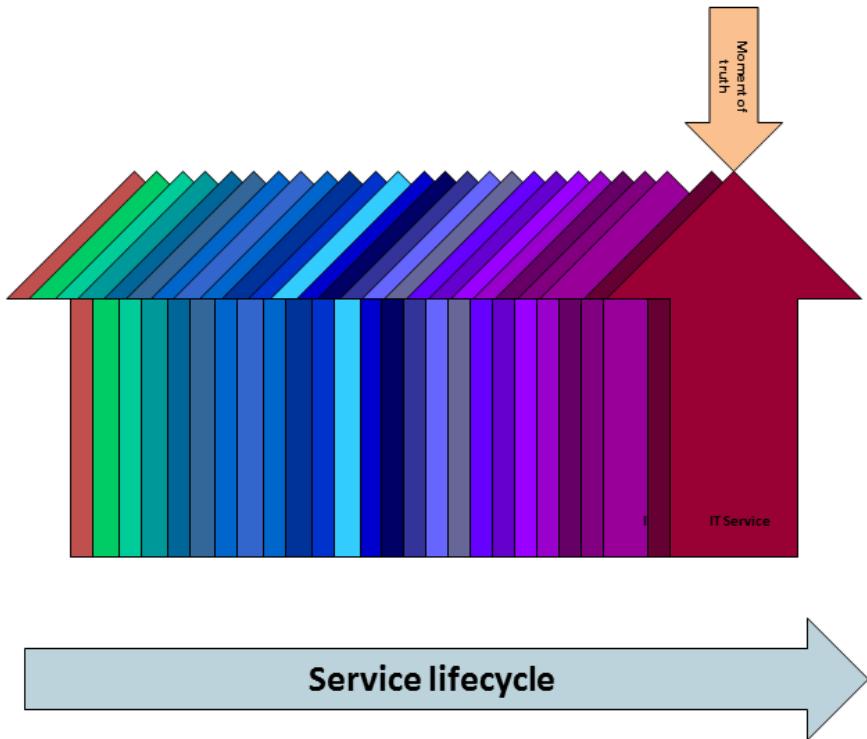


Figure XX: The IT service lifecycle

[add full 2-axis treatment, and discussion of production vs product design]

Defining consumer, customer, and sponsor

In understanding IT value, it is essential to clarify the definitions of user, customer, and sponsor. Sometimes, the user **is** the customer. But more often, the user and the customer are different, and one may additionally need to distinguish the role of system or service sponsor.

The following definitions may help

- The **consumer** (sometimes called the **user**) is the person actually interacting with the IT-based service.
- The **customer** is a source of revenue for the service. If the service is part of a profit center, the customer is the person actually purchasing the product (e.g. demand deposit banking.) If the service is part of a cost center (e.g. an HR system), the customer is best seen as an internal executive, as the actual revenue-producing customers are too far removed.
- The **sponsor** is the person who authorizes and controls the funding used to construct and operate the service.

Depending on the service type, these roles can be filled by the same or different people. Here are some examples:

Example	Consumer	Customer	Sponsor	Notes
Online banking		Bank account holder	Vice president, consumer banking	Customer-facing profit center with critical digital component
Online restaurant reservation application	Restaurant customers	Restaurant owners	Product owner	Profit-making digital product
Enterprise human resources application	HR analyst		Vice president, HR	Cost center funded through corporate profits
Online video streaming service	End video consumer (e.g. any family member)	Streaming account holder (e.g. parent)	Streaming video product owner	Profit-making digital product
Social traffic application	Driver	Advertiser, data consumer	Product owner	Profit-making digital product

So, who paid for our user's enjoyment? The bank and restaurant both had clear motivation for supporting a better online experience, and people now expect that service organizations provide this. The bank experiences less customer turnover and increased likelihood that customers add additional services. The restaurant sees increased traffic and smoother flow from more efficient reservations. Both see increased competitiveness.

The traffic application is a somewhat different story. While it is an engineering marvel, there is still some question as to how to fund it long term. It requires a large user base to operate, and yet end consumers of the service are unlikely to pay for it. At this writing, the service draws on advertising dollars from businesses wishing to advertise to passersby, and also sells its real-time data on traffic patterns to a variety of customers, such as developers considering investments along given routes.

This last example well illustrates the maxim that "if you don't know how the product is making money, you are the product."

Collaboration notes

*todo: let's put in basic impact mapping here see
https://en.wikipedia.org/wiki/Outside%20in_software_development*

-ctb

IT exists in a social context

Like the proverbial fish that doesn't understand water (because water it all it knows), we may lose sight of the laws and customs that enable us to use computers in the ways covered in this chapter.

For example, the ability for banks to hold money as electronic bits on a computer is rooted in the earliest history of banking and the emergence of centralized settlement and clearing mechanisms. Cell phone companies rely on international treaties, and national laws and regulations allocating radio spectrum.

The existence of physical voice and data connectivity relies on laws supporting utility easements and rights of way, and even treaties such as the Law of the Sea (how is it that undersea cables remain unmolested?).

In short, without a social infrastructure, social and collaborative technology has no meaning.

consider discussion of organizational mission, foreshadow chapter 10 discussion on mission/vision/principles/policies/procedures

Understanding digital context

As you consider embarking on a journey of IT value, you need to orient to your surroundings and create an initial proposal or plan for how you will proceed. If you are actually a startup, you need a business plan. If you are working as an intrapreneur in a larger organization, you will still need some kind of formal proposal. This section describes some tools and thinking approaches that may be useful at this very earliest stage. There are more focused, product-specific techniques in the chapter section on [product discovery techniques](#).

Market facing, supporting, back office

In Chapter 1 we discussed the question of "who pays/who benefits" for the service, [proposing that](#) the service consumer, the service customer, and the service sponsor might be three distinct roles (sometimes collapsing into 2 or 1 individual.)

We see this again in how we can categorize the "customers" of IT services and systems. Roughly, such services can be:

- Directly market- and consumer-facing (e.g. Facebook), to be used by external consumers and paid for by either them or closely associated customers (e.g. Netflix, or an online banking system).
- Customer "supporting" systems, such as the online system that a bank teller uses when interacting with a customer. Customers do not interact directly with such systems, but customer-facing representatives do, and problems with such systems may be readily apparent to the end customer.

- Completely "back-office" systems (HR, payroll, marketing, etc)

Note however, that (especially in the current digitally transforming market) a service previously thought of as "back office" (when run internally) becomes "market-facing" when developed as a profit-seeking offering. For example, an HR system built internally is "back office," but Workday is a directly market-facing product, even though the two services may be similar in functionality.

In other words, it's all relative.

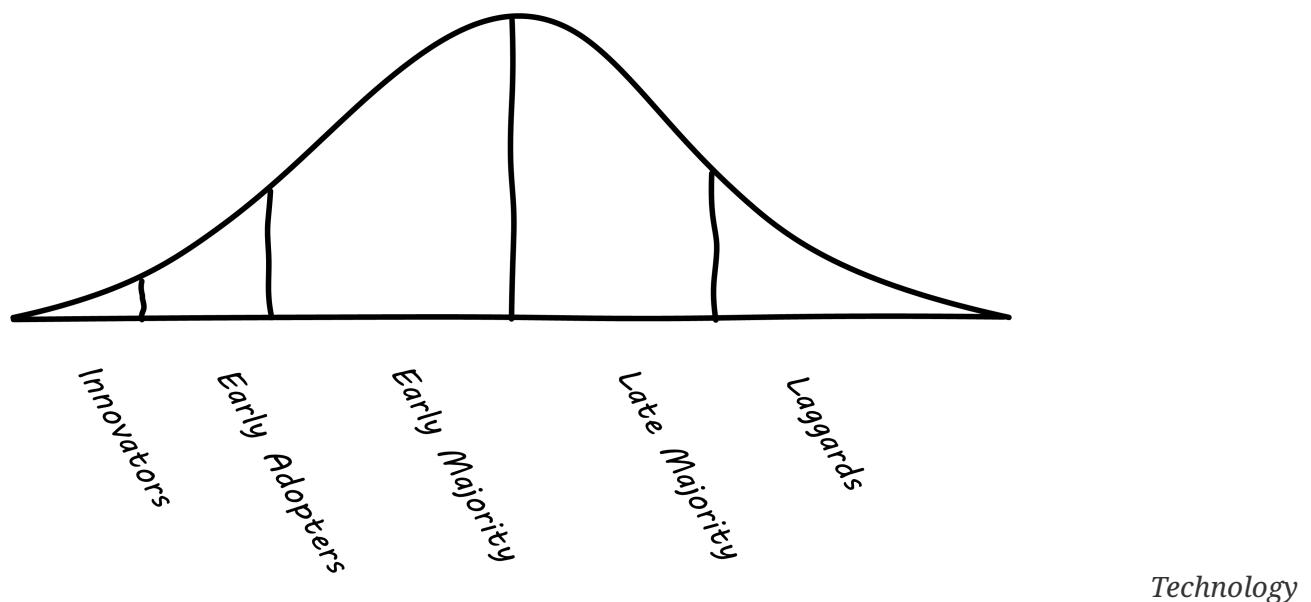
Especially when products are **not** market facing, we start to run into the problem of distinguishing **discovery versus design**, as we discuss below.

Bente notes that Broadbent and Kitzis (2005) distinguish three fundamental business outlooks- fighting for survival, maintaining competitiveness, breaking away

Diffusion theory and other approaches

As you start to think about digital value, you must think about the context for your startup or product idea. What is the likelihood of its being adopted? Is it part of a broader "movement" of technological innovation? Where is the customer base in terms of its willingness to adopt the innovation? A well known approach is the idea of the "diffusion theory," first researched by Everett Rogers and proposed in his *Diffusion of Innovations*, [\[Rogers2003\]](#).

Rogers' research proposed the idea of "Adopter Categorization on the Basis of Innovativeness," with a well-known graphic:



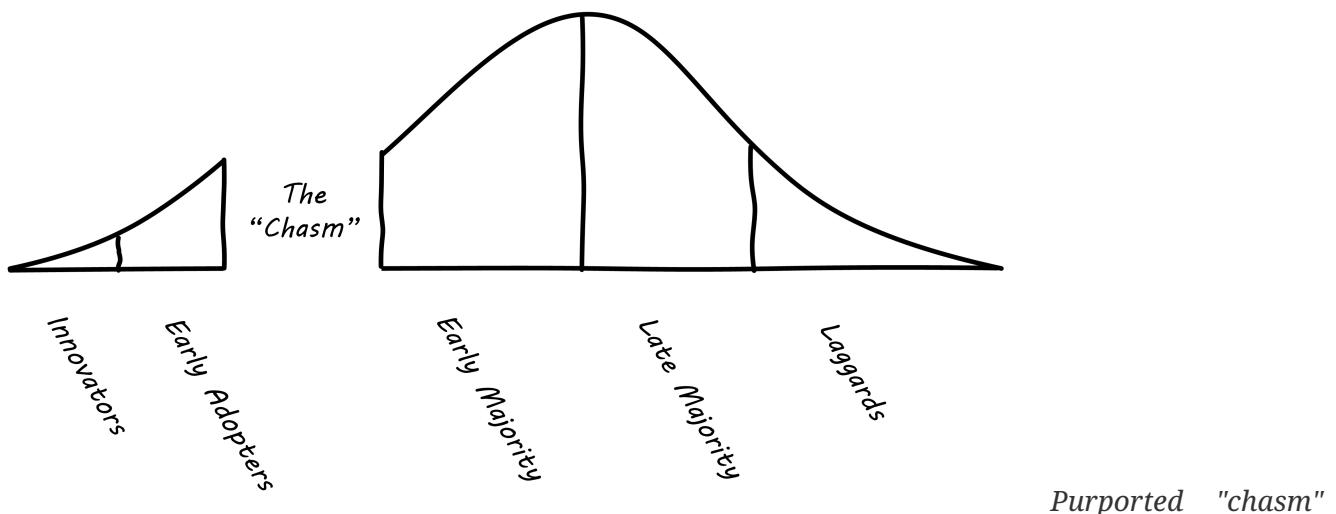
Technology adoption categories, similar to [Rogers2003], figure 7-3, page 281.

Rogers went on to characterize the various stages:

- Innovators: Venturesome risk-takers
- Early adopters: Opinion leaders
- Early majority: Deliberative, numerous

- Late majority: Skeptical, also numerous
- Laggards: Traditional, isolated, conservative

The above figure was popularized in the following variation by Geoffrey Moore [[Moore2014](#)] in his bestseller *Crossing the Chasm*:



between adopter categories, similar to [[Moore2004](#)], page 21.

You'll see this diagram often, but you should be aware that Rogers himself (the person who has researched the data) says that (p.282):

Past research shows no support for this claim of a "chasm" between certain adopter categories. On the contrary, innovativeness, if measured properly, is a continuous variable and there are no sharp breaks or discontinuities between adjacent adopter categories (although there are important differences between them).

The idea of technology diffusion frames the problem for us, but we need more. Steve Blank, in his influential book *The Four Steps to Epiphany* [[Blank2013](#)], argues there are four categories for startups (p.31):

- Startups that are entering an existing market
- Startups that are creating an entirely new market
- Startups that want to re-segment an existing market as a low-cost entrant
- Startups that want to re-segment an existing market as a niche player

Understanding which category you are attempting is critical, because "the four types of startups have very different rates of customer adoption and acceptance."

Another related and well known categorization of competitive strategies comes from Michael Treacy and Fred Wiersma [[Treacy1997](#)]:

- Customer intimacy
- Product leadership
- Operational excellence

It is not difficult to categorize well known brands in this way:

Customer intimacy	Product leadership	Operational excellence
Nordstrom Home Depot	Apple Nike	Dell Computer Wal-Mart

However, deciding which strategy to pursue as a startup may require some experimentation.

Business discovery frameworks

Startups that survive the first few tough years do not follow the traditional product-centric launch model espoused by product managers or the venture capital community...In particular, the winners invent and live by a process of customer learning and discovery. I call this process “Customer Development,” a sibling to “Product Development,” and each and every startup that succeeds recapitulates it, knowingly or not.

— Steve Blank, *The Four Steps to Epiphany*

Let's start with two well known approaches that can help you bridge from an understanding of your product context, to an effective vision for building and sustaining a product:

- Eric Ries' Lean Startup
- Alexander Osterwalder's Business Model Canvas

Lean Startup

The goal of a startup is to figure out the right thing to build— the thing customers want and will pay for— as quickly as possible. In other words, the Lean Startup is a new way of looking at the development of innovative new products that emphasizes fast iteration and customer insight, a huge vision, and great ambition, all at the same time.

— Eric Ries, *The Lean Startup*

Lean Startup is a philosophy of entrepreneurship developed by Eric Ries [Ries2011]. It is not specific to information technology; rather, it is broadly applicable to all attempts to understand a product and its market. (Remember according to our [definition of product management](#) that a workable market position is essential to any product.)

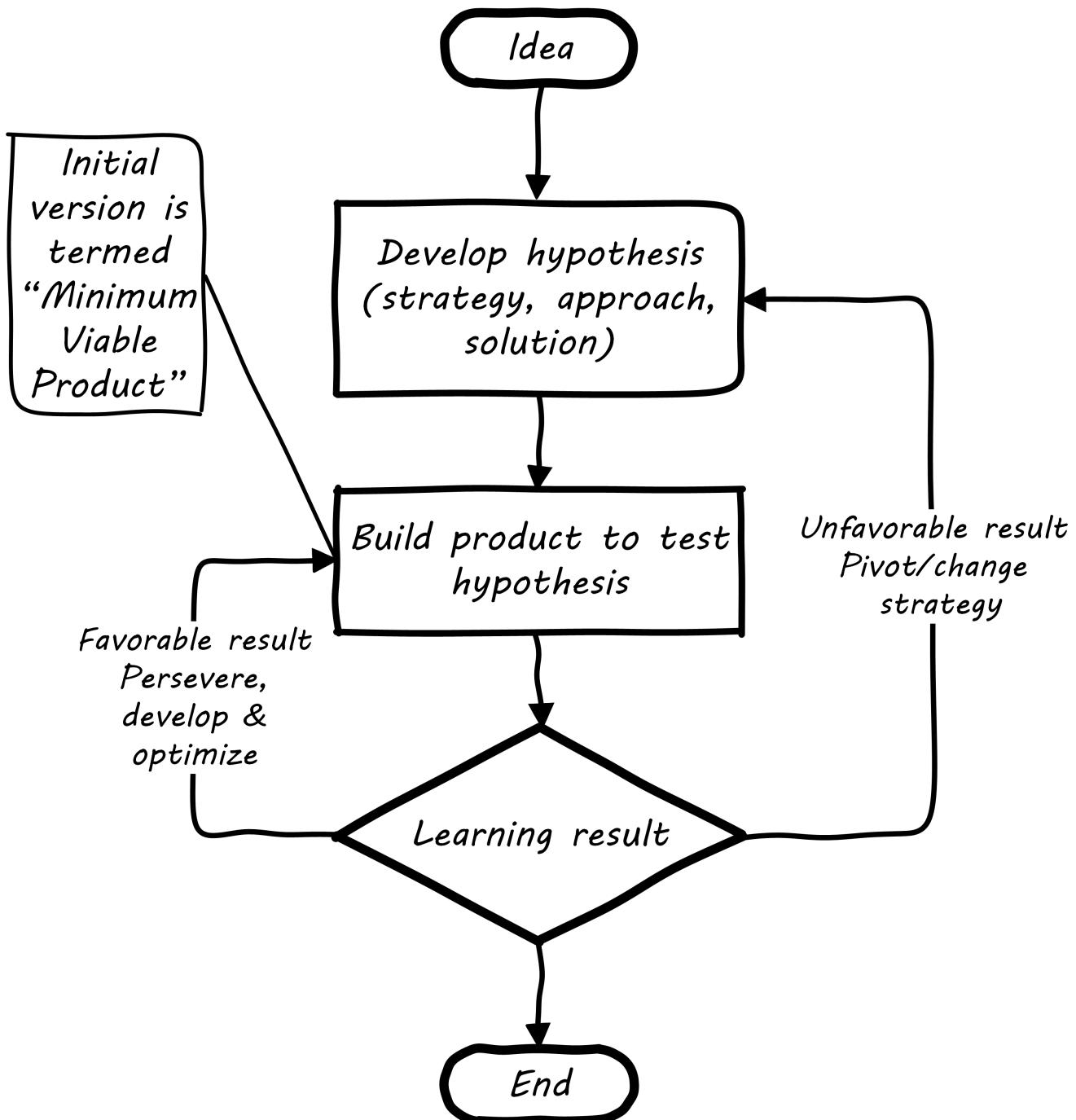
The idea of the Lean Startup has had profound influence on product design, including market-facing and even internal IT systems. It is grounded in Agile concepts such as:

"Do the simplest thing that could possibly work."

Lean Startup calls for an iterative, "Build-Measure-Learn" cycle. Repeating this cycle frequently is the essential process of building a successful startup (whatever the digital proportion).

- Develop an idea for a Minimum Viable Product (MVP)
- Measure its effectiveness in the market (internal/external)
- Learn from the experiment
- Decide to persevere or pivot (change direction while leveraging momentum)
- New idea development, evolution of MVP

Flowcharts such as this are often seen to describe the Lean Startup process:



Business model canvas

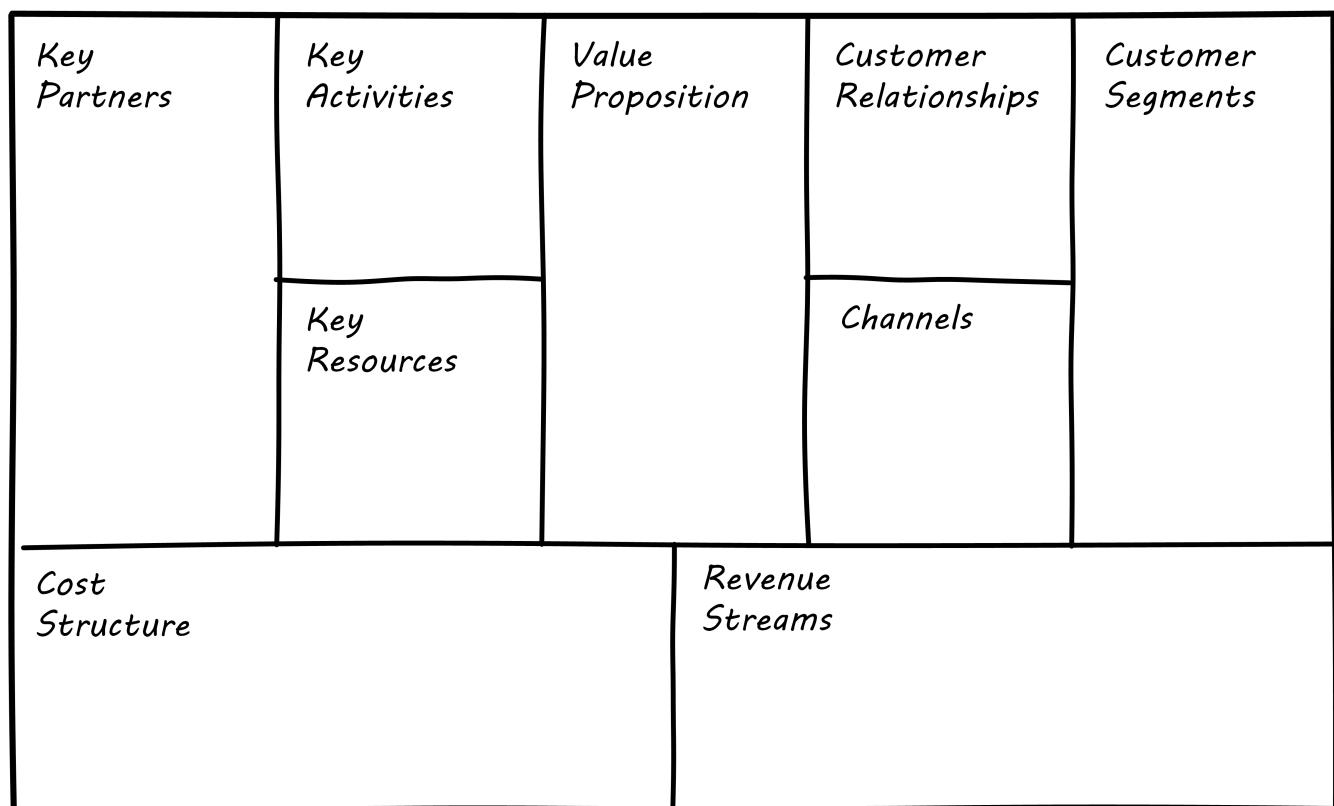
One recent book that's been influential among entrepreneurs is Alex Osterwalder's *Business Model Generation* [[Osterwalder2010](#)].

This book is perhaps best known for introducing the concept of the Business Model Canvas, which it defines as "A shared language for describing, visualizing, assessing, and changing business models."

The Business Model Canvas uses 9 major categories to describe the business model:

- Key Partners
- Key Activities
- Value Proposition
- Customer Relationships
- Customer Segments
- Key Resources
- Channels
- Cost Structure
- Revenue Streams

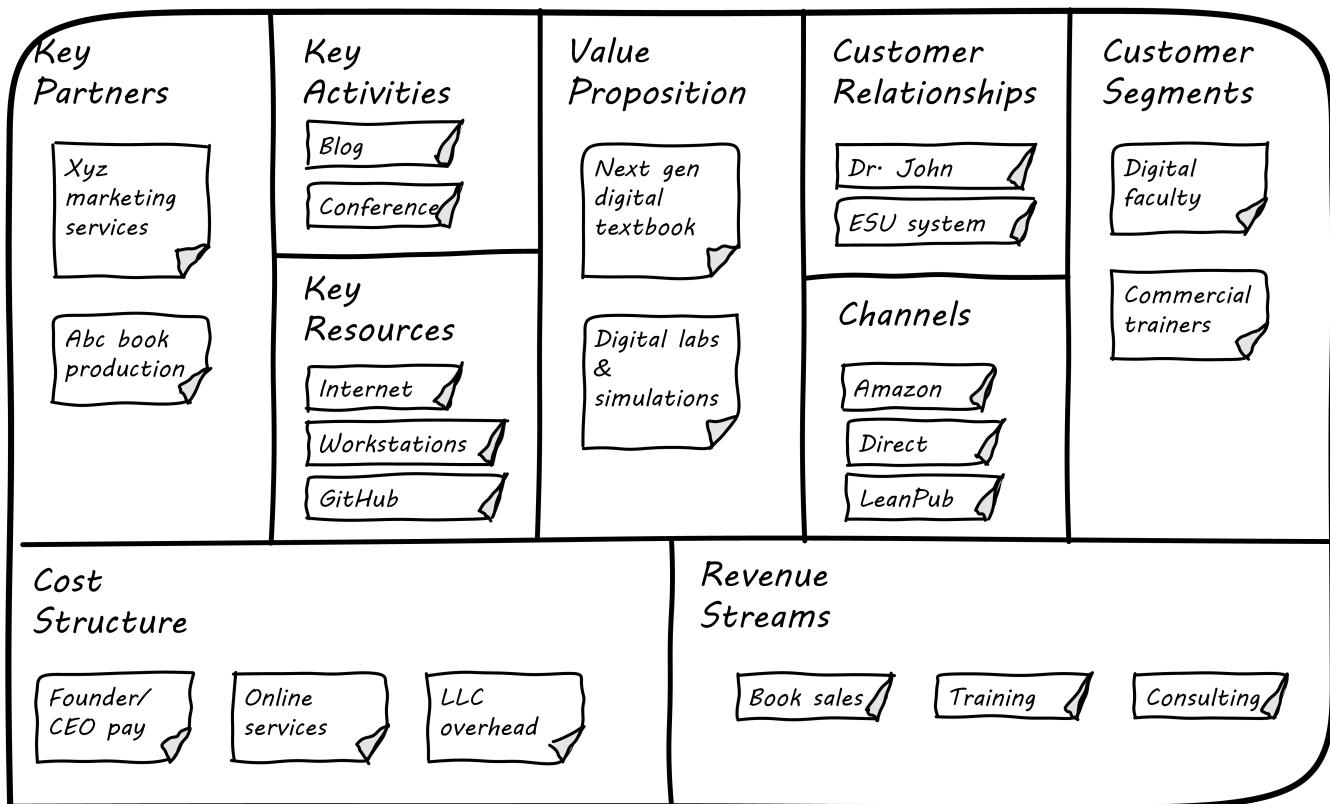
and suggests they be visualized in this manner:



similar to [[Osterwalder2010](#)], p. 44

The canvas is then used in collaborative planning, e.g. as a large format wall poster where the business team can brainstorm, discuss, and fill in the boxes (e.g., what is the main "Value

Proposition"? Mobile bank account access?):



A rough approximation of the author's business model on the Business Model Canvas

Osterwalder and his colleagues, in *Business Model Generation* and the followup *Value Proposition Design* [Osterwalder2014], suggest a wide variety of imaginative and creative approaches to developing business models and value propositions, in terms of patterns, processes, design approaches, and overall strategy.

===== The enterprise context

We will discuss in Chapter 12. For now, assume that your role and priorities are defined by some strategic planning and investment activities. You need to understand the mission to the best of your abilities, even as an individual contributor. This book will take you through the journey of understanding how such priorities are set.

important - commentary on Agile Product Roadmap - arrow
<https://twitter.com/ScrumDan/status/786729964422909953>

when do they become explicit?

Current digital business trends

Apps

Big Data

Internet of Things

Platforms not products

Bring in Rouser: Digital Strategy: A Guide to Digital Business Transformation, Rogers: The Digital Transformation Playbook: Rethink Your Business for the Digital Age, and Westerman various.

"Post-industrialism turns brand management into a digital conversation between a company and its customers." <<Sussna2015>>

Conclusion

In this chapter, we discussed the basic questions of information technology value and how it is experienced and developed. Through the mechanism of a hypothetical modern IT user, we covered (at a **very** high level) the necessary ingredients of the IT experience. We considered the user's moment of truth, and the massive IT complexity that sustains it. We also discussed a high level lifecycle model for IT applications and services, and explored some initial definitions for user, customer, and sponsor - critical distinctions to make in an age of digital transformation.

As you proceed into the course, the key takeaway from this chapter is "why IT?" Why do people need it, and how is it valuable? That should always remain at the top of your mind as you proceed in your IT education.

Discussion questions

1. Discuss: How does information technology contribute to your enjoyment of life and experiences of value?
2. Read [Apps Are Wrecking Mom-and-Pop Pizza Shops](#) and discuss whether "IT matters" to the local pizzerias.
3. Read the Wikipedia articles on [mainframe computing](#) and [Amazon Web Services](#) and discuss with your team. What has changed in computing? What remains the same?

Research & practice

1. Go to any popular online service (Facebook, Netflix, Flickr, etc.) How would you describe the "moments of truth" or value experiences these sites offer users? There may be several.
2. On your own or with a team, develop an idea for an IT-based product you could take to market. Present to the class.
3. (Continued) Who are the user, customer, and sponsor of your product? (See previous discussion in chapter.)

Further reading

Books

- [Code](#), Charles Petzold
- [The Information](#), James Gleick
- [The Lean Startup](#), Ries

Articles

- [definitions of IT stack](#) (follow links)
- [Working the Land and the Data](#)

Picture credits:

<https://pixabay.com/en/woman-sitting-counter-phone-828888/>

<https://pixabay.com/en/friends-celebration-dinner-table-581753/>

<https://www.flickr.com/photos/iicd/5348620457/>

2nd draft notes

Incorporate insights from The-Nerdery_Product-Development_ebook.pdf
Entrepreneurs tell a story entrepreneurs tell a story <https://silvialisam.com/chris-sacca-s-639f0f5f3b9c#.j6sljdp0x> (first section??)

Cynevin. Complicated vs complex, what is computable.

Chapter 2: Infrastructure Management

Introduction to Chapter 2

Collaboration notes

There is a wealth of material covering IT infrastructure in far more detail than possible here. We don't want to rewrite Allspaw or Limoncelli. Curating the most useful aspects for a survey text is the challenge.

Note that Chapter 6 is dedicated to IT operations.

-ctb

As mentioned in the section introduction, you cannot start developing a product until you decide what you will build it with. (You may have a difficult time writing an app for a mobile phone if you choose the COBOL programming language!) You also need to understand something of how computers are operated, enough so that you can make decisions on how your system will run. Most

startups choose to run IT services on infrastructure owned by a Cloud provider, but there are other options. Certainly, as you scale up, you'll need to be more and more sophisticated in your understanding of your underlying IT services.

Configuring your base platform is one of the most important capabilities you will need to develop. You'll never stop doing it. The basis of modern configuration management is [version control](#), which we cover here.

This is one of the more technical chapters. Supplementary reading may be required for those completely unfamiliar with computing. See [Assumptions of the Reader](#) for notes on the book's approach.

Chapter summary

- Defining "IT infrastructure"
 - How an IT system **runs**, as opposed to what it **does**
 - The ingredients (OS, language, services)
 - Physical vs virtual
 - Cloud systems
 - Servers vs containers vs services
- IT infrastructure from a management point of view
 - Choosing an infrastructure platform
- Operations overview (brief; covered in depth in [Chapter 6](#))
- Automation and infrastructure as code
 - Policy-based approaches
- The critical importance of version control
- Scalability and infrastructure architecture - looking ahead

Learning objectives

- Understand fundamental principles of operating computers as infrastructure for a service
- Understand Cloud as a computing option
- Understand basic principles of "infrastructure as code"
- Understand the importance and basic practices of version control and why it applies to infrastructure management

Infrastructure overview

In the previous chapter, you were introduced to the concept of a Moment of Truth, and in the final exercises asked to think of a product idea. Some part of that product requires writing software, or at least configuring some IT-centric system. (IT being defined as in [Chapter 1](#).)

Moment of Truth & Minimum Viable Product

NOTE A Minimum Viable Product is required to deliver an initial Moment of Truth. You need infrastructure capable of supporting your Minimum Viable Product's idea.

You presumably have some resources (time and money). It's Monday morning, you have cleared all distractions, shut down your Twitter and Facebook feeds, and are ready to start building.

Not so fast.

Before you can start writing code, you need some kind of a platform. It's hard to build before you decide on your tools.

You need to decide what language you are going to write in, or what framework you are going to configure, and how that effort is going to result in an operational system capable of rendering IT services.

You are probably swimming in a sea of advice and options regarding your technical choices. In previous decades, books such as this might have gone into the specifics of particular platforms: mainframe vs. minicomputers, COBOL vs Fortran, Windows vs Unix, etc.

At this writing, JavaScript is a leading choice, in conjunction with various frameworks and NoSQL options (e.g. the MEAN stack, for MongoDB, Express, Angular, and Node.js), but millions of developers are still writing Java and .Net, and Ruby and Python have significant followings. Linux is arguably the leading platform, but commercial Unix and Microsoft platforms are still strong. And, periodically it's reported that the majority of the world's transactions **still** run on [COBOL-based systems](#).

However, in the past few years, some powerful infrastructure concepts have solidified that are independent of particular platforms:

- Automation and “infrastructure as code”
- The centrality of source control
- The importance of package management
- Policy-based infrastructure management

(We'll get to test-driven development, pipeline automation & DevOps in the next chapter.)

This might seem like a detour - you are in a hurry to start writing code! But industry practice is clear. You check your code into source control from Day One. You define your server configurations as recipes, manifests, or at least shell scripts, and check those definitions into source control as well. You keep track of what you have downloaded from the Internet and what version of stuff you are using, through package management (which uses different tools than source control). Always downloading the “latest” package from its upstream creator might seem like the way to stay current, but it will kill you when stuff works on one server but not on another.

So, you need to understand a few things and make a few decisions that you will be living with for a while, and will not be easily changed.

What is Infrastructure?

Infrastructure is a tricky word. Google defines it thus:

The basic physical and organizational structures and facilities (e.g., buildings, roads, and power supplies) needed for the operation of a society or enterprise.

In general, it connotes the stuff behind the scenes, the things you need but don't want to spend a lot of time thinking about.

We will spend a lot of time examining what we mean by "infrastructure" in this book, as it is fundamental to understanding the "business of IT."

This book defines "IT infrastructure" recursively as "the set of IT concerns that are of particular interest to IT."

- An application or business service is consumed by people who are NOT primarily concerned with IT. For example, a customer-facing online banking service is consumed by end users.
- An IT infrastructure service is a service consumed by other IT-centric teams and capabilities. For example, a database or a load balancing service is consumed by other IT teams.

IT infrastructure is one form of infrastructure. Other kinds of infrastructure might include mechanical, electrical, and plant investments (ME & P). IT infrastructure, like IT itself, is defined by its fundamental dependence on information and computing theory [link Chapter 1].

Today's application becomes tomorrow's infrastructure. Forty years ago, building an "application" would have included building its database, perhaps even its file management. This was rightly determined to be a general-case problem that could be the basis for commodity software, and so companies like Oracle were born.

NOTE

Operating systems took on more and more functionality, technology became more and more reliable and easily configured, and once unique (and competitively differentiating) functionality became commoditized and more and more deeply buried in the "stack": the IT infrastructure.

Basic IT infrastructure concepts

There are many books (some in the Further Reading section for this chapter) on all aspects of IT infrastructure, which is a broad and deep topic. Our discussion of it here has to be high level, as appropriate for a survey course.

We've established [what we mean by IT generally](#). Pragmatically, there are three major physical aspects to "IT infrastructure" relevant to the practitioner:

- Computing cycles (sometimes called just "compute")

- Memory & storage (or "storage")
- Networking & communications (or "network")

We will discuss a variety of subsidiary concerns and concepts, but those are the big three.

Compute is the resource that performs the rapid, clock-driven digital logic that transforms data inputs to outputs.

TIP For a beginner-level introduction to the world of digital logic at its most fundamental, see [Code](#), by Charles Petzold.

If we have a picture of our friend, and we use a digital filter to adjust the brightness, that is an application of compute power. The picture is made up of "pixels" which are nothing but numbers representing the color of some tiny square of the picture. Each of those numbers needs to be evaluated and the brightness adjusted. There may be millions in a single image.

Add illustration of bitmap

For example, let's say that the pixel values range from 1 to 10, with 1 being the darkest and 10 being the lightest (in reality, the range is much larger.) We might tell the computer:

```
Look at a pixel  
If it is between 0 and 3 add 2  
If it is between 4 and 6 add 1  
Move to a new pixel  
Repeat the above 4 lines until all pixels are done
```

At a more realistic level, the process might be executed as a batch on a workstation, for hundreds or thousands of photos at a time.

Computers process instructions at the level of "true" and "false," represented as [binary](#) "1s" and "0s." Because humans cannot easily understand binary data and processing, higher-level abstractions of [machine code](#) and [programming languages](#) are used.

Computer processing is not free. Moving data from one point to another - the fundamental [transmission of information](#) - requires matter and energy, and is bound up in physical reality and the [laws of thermodynamics](#). (This despite claims like, "information wants to be free.") In the real world, even running the simplest calculation has physical and therefore economic cost, and so we must pay for computing.

IMPORTANT If these concepts are strange to you, spend some time with the suggested Wikipedia articles, or otherwise researching the topics. Wikipedia in the area of fundamental computer concepts is generally accurate.

Storage But where did the picture come from? The data comprising the pixels needs to be stored somewhere. Sometimes you will hear the technical term "persisted." The combined set of pixels and their precise values can be termed the "state" of the photograph; the digital logic of the filter alters

the state, and also needs to save this new state somewhere (otherwise it will be lost).

Many technologies [have been used for digital storage](#). Increasingly, the IT professional need not be concerned with the physical infrastructure used for storing data. As we will cover in the next section, storage increasingly is experienced as a virtual resource, accessed through executing programmed logic on Cloud platforms. "Underneath the covers" the Cloud provider might be using various forms of storage, from RAM to solid state drives to tapes, but the end user is, **ideally**, shielded from the implementation details (part of the definition of a service).

However, it is important to understand that in general, storage follows a [hierarchy](#). Just as we might "store" a document by holding it in our hands, setting it on a desktop, filing it in a cabinet, or archiving it in a banker's box in an offsite warehouse, so computer storage also has different levels of speed and accessibility:

- On-chip registers and cache
- Random-access memory (RAM), aka "main memory"
- On-line mass storage, often "disk"
- Off-line mass storage, e.g. "tape"

If this is unfamiliar, see Wikipedia or research on your own; you should have a basic grasp of this issue.

Network We can change the state of some data, or store it. We also need to move it. This is the basic concern of [networking](#), to transmit data (or information) from one location to another. If you use your cell phone to look up your bank balance, there is network involved - otherwise, how did the data get from the bank's computer in New Jersey to your cell phone in Minnesota?

Networking is fairly intuitive; you may be familiar with coaxial cables for TV, or telephone lines strung from pole to pole in many areas. However, like storage, there is also a hierarchy of networking:

- Intra-chip pathways
- [Motherboard](#) and [backplane](#) circuits
- [Local area networks](#)
- [Wide area networks](#)
- [Backbone networks](#)

Like storage, networking as a service increasingly is independent of implementation. The developer uses programmatic tools to define expected information transmission, and (again **ideally**) need not be concerned with the specific networking technologies or architectures serving their needs.

IMPORTANT

Eventually, scale matters. As your IT service's usage increases, you will inevitably find that you need to start caring about technical details such as storage and network architecture.

The implementation decisions made by you and your service providers may become inefficient for the particular "workload" your product represents, and you will need to start asking questions. This will be discussed in more depth in the Cloud chapter.

The brief technical writeup, [Latency Numbers Every Programmer Should Know](#) can help you start thinking about these issues.

Choosing infrastructure



Tower-style servers on a rack.

<https://www.flickr.com/photos/sflaw/78981814>, Creative Commons Attribution-ShareAlike 2.0 Generic (CC BY-SA 2.0)

The following is written as if you are a decision maker in the early stages of conceiving a product. If you are an individual contributor in a large enterprise, or even a newcomer to an established product team, these decisions will likely have been made for you.

NOTE

But at some point, **someone** had to go through this decision process, before anything could be developed.

There is ferocious turbulence in the IT infrastructure market. Cloud technologies, DevOps, various platform wars... As an entrepreneur, you need to understand what technical trends matter and what are more of same.

In particular, you will need to make some level of commitment to your technical architecture. And at some point you **WILL** be asked, “You’re still using that old technology?”

As a startup, you have a couple initial decisions to make regarding infrastructure and tools:

- What is my vision for bringing my product to the world?
- What toolset should I use to forward this vision?

As a startup, it would seem likely that you would use a commodity cloud provider. This text is based on this assumption (physical IT asset management will be discussed in Sections 3 and 4), but is there any reason why that would not work for you?

The two questions are related. If you want to develop on a LAMP stack, you need a Cloud provider that will support you in this. While most are very flexible, you will need to consider the specific support levels they are offering; a provider that supports the platform (and not just the operating system) might be more valuable, but there may be higher costs and other tradeoffs.

There is a near-infinite amount of material, debate, discussion, books, blogs, lists, and so forth concerning choice of language and platform. Exploring this area is not the purpose of this book.

However, this book has certain assumptions:

- Your system will be built, at least in part, with some form of programming language which is human-readable and compiled or interpreted into binary instructions.
- Your system will run on general purpose digital computers using well known technologies.
- Your computing environment is networked in a standard way.
- You use the concept of a software pipeline, in which new functionality is developed in a scope distinct from what is currently offered as your product/service.
- New functionality moves through the pipeline at significant volumes and velocity and you are concerned with optimizing this overall flow.

Infrastructure automation enables rapid iterations and scaling.

IMPORTANT

Iterative development and rapid scaling, while possible, was often more difficult with earlier, less automated technical platforms.

Experienced IT professionals may be jumping up and down at this point, saying “How can you be thinking about infrastructure before you have gone deeply into requirements?”

Let’s be clear, in defining a product (Chapter 1) you already have started to think about requirements, although we have not yet started to use that term. The Minimum Marketable Feature is an initial statement of requirements from which you should be able to infer at least initial toolset and platform requirements. Here to get you started are the major players as of fall 2015:

Stack 1 (Enterprise Java)	Stack 2 (Microsoft)	Stack 3 (LAMP)	Stack 4 (MEAN)
Java	C#	PHP	Javascript, Express & Angular
Oracle DB	MS SQL Server	MySQL	MongoDB
Websphere	.Net	Apache Web Server	NodeJS
Commercial Unix	Microsoft Windows	Red Hat Linux	Ubuntu Linux

Ruby on Rails is another frequently-encountered platform.

There is a good reason, however, why you should not spend too much time “analyzing” before you make platform decisions. The reality is that you cannot know all of the factors necessary to make a perfect decision, and in fact the way you will learn them is by moving forward and testing out various approaches.

So, choose Ruby on Rails, or LAMP, or MEAN, and a hosting provider who supports them, and start. Do not fall into analysis paralysis. But **be critical of everything** especially in your first few weeks of development. Ask yourself:

- Can I see myself doing things this way for the next year?
- Will I be able to train people in this platform?
- Will this scale to a bigger codebase? Higher performance? Faster throughput of new features?

If you become uncomfortable with the answers, you should consider alternatives.

The technical "spike"

TIP

Scrum-based development uses the concept of a “spike” to represent effort whose outcome is not shippable product, but rather research and information. Consider thinking of your infrastructure selection process in this way.

From "physical" compute to Cloud

According to the [National Institute for Standards and Technology](#), Cloud is:

a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Before Cloud, people generally bought computers of varying sizes and power ratings to deliver the IT value they sought. With Cloud services, the same compute capacity can be rented or leased by the minute or hour, and accessed over the Internet.

There is much to learn about Cloud computing. In this introduction, we will discuss the following aspects:

- Virtualization
- Managed hosting
- What differentiates Cloud
- The various kinds of Cloud services

Virtualization

[Virtualization](#), for the purposes of this section, starts with the idea of a computer within a computer. [6: it has applicability to storage and networking as well but we will skip that for now] In order to understand this, we need to understand a little bit about operating systems and how they relate to the physical computer.

Assume a simple, physical computer such as a laptop:



When the laptop is first turned on, the operating system (OS) loads; the OS is itself software, but is able to directly control the computer's physical resources: its CPU, memory, screen, and any interfaces such as WiFi, USB, and Bluetooth.

The operating system (in a traditional approach) then is used to run "applications" such as Web

browsers, media players, word processors, spreadsheets, and the like. Many such programs can also be run as applications within the browser, but the browser still needs to be run as an application.

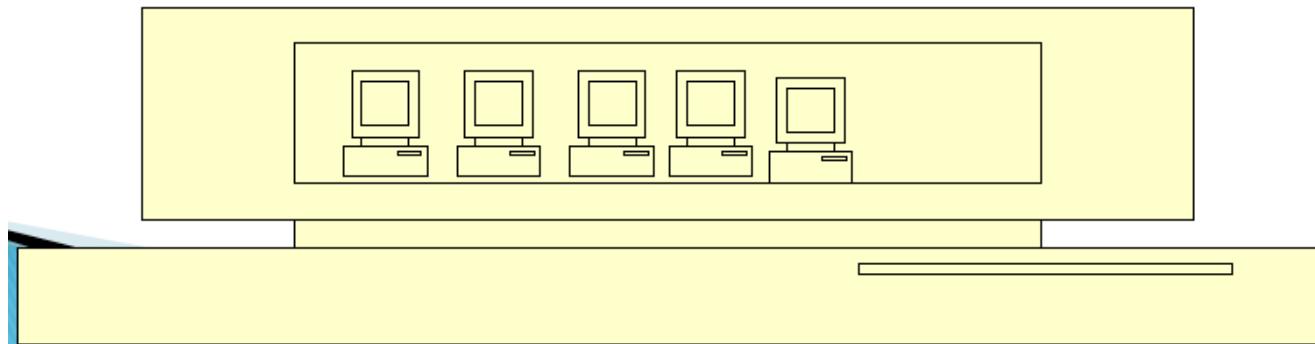


Figure XX. Virtualization is computers within a computer.

In the simplest form of virtualization, a specialized application known as a hypervisor is loaded like any other application. The purpose of this hypervisor is to emulate the hardware computer in software. Once the hypervisor is running, it can emulate any number of "virtual" computers, each of which can have its own operating system. The hypervisor mediates the [virtual machine](#) (VM) access to the actual, physical hardware of the laptop; the VM can take input from the USB port, and output to the Bluetooth interface, just like the master OS that launched when the laptop was turned on.

You can experiment with a hypervisor by downloading [Virtualbox](#) (on Windows, Mac OS, or Linux) and using [Vagrant](#) to download and initialize a Linux virtual machine.

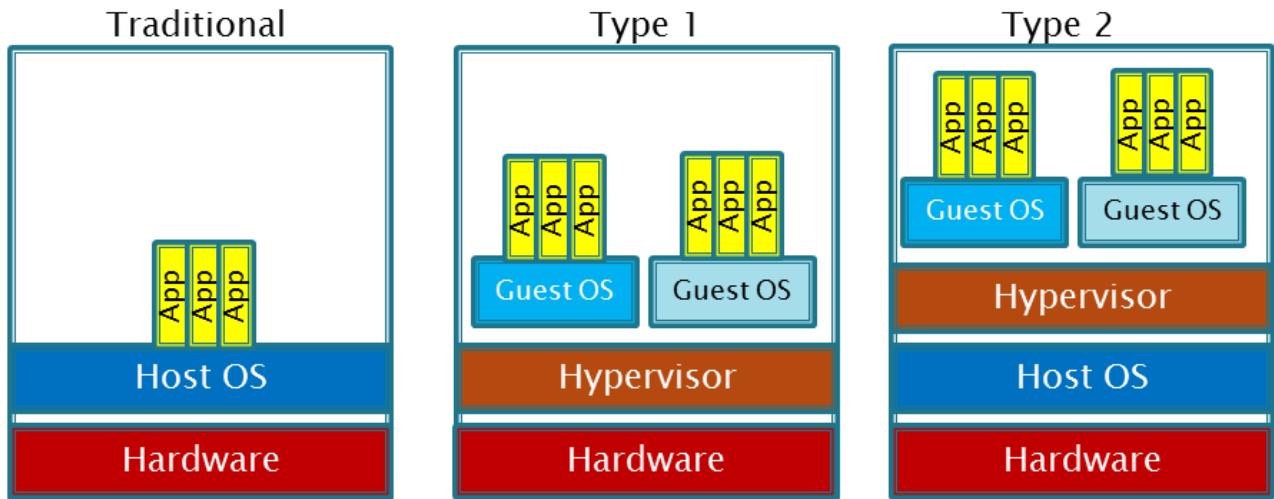
TIP

You'll probably want at least 4 gb of RAM on your laptop and a gigabyte of free disk space, at the bare minimum.

There are two different kinds of hypervisors. The example we just discussed was an example of a Type 2 hypervisor, which runs on top of a host operating system. In a Type 1 hypervisor, a master host OS is not used; the hypervisor runs on the "bare metal" of the computer and in turn "hosts" multiple VMs.

Paravirtualization, e.g. containers, is another form of virtualization found in the marketplace. In a paravirtualized environment, a core operating system is able to abstract hardware resources for multiple virtual guest environments without having to virtualize hardware for each guest. The benefit of this type of virtualization is increased I/O efficiency and performance for each of the guest environments.

However, while hypervisors can support a diverse array of virtual machines with different operating systems on a single computing node, guest environments in a paravirtualized system generally share a single OS.

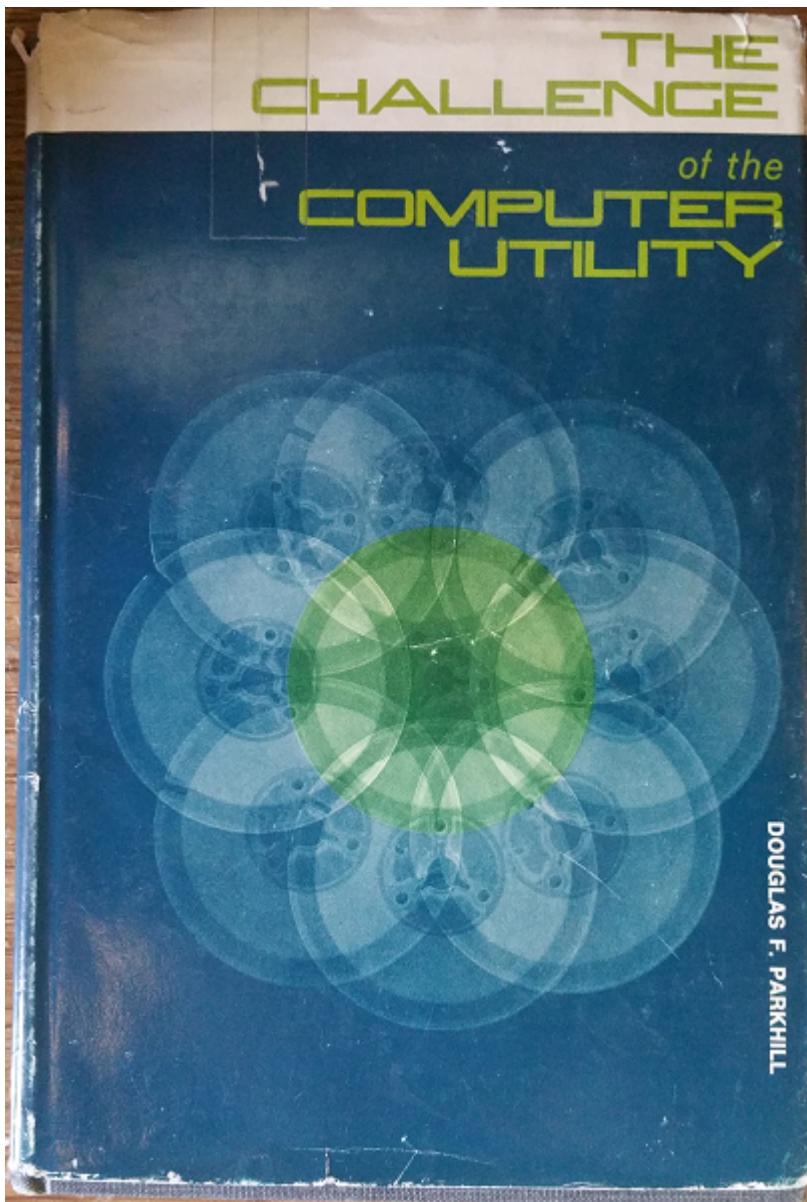


Why is virtualization important?

While the conceptual idea of Cloud and utility computing was foreseeable fifty years ago, it took many years of hard-won IT evolution to support the vision. Reliable hardware of exponentially increasing performance, robust open-source software, Internet backbones of massive speed and capacity, and many other factors converged towards this end.

Cloud computing has its uses even on a laptop (as you will see if you follow certain paths in the labs.) However, it is in larger industrial uses where it has the greatest utility.

The idea of running IT as a utility service goes back at least to 1965 and the publication of *_The Challenge of the Computer Utility*, by Douglas Parkhill.



The author's copy of Parkhill's book

Virtualization was predicted in the earliest theories that led to the development of computers. Turing and Church realized that any general purpose computer could emulate any other. Virtual systems have existed in some form since [at latest 1967](#) - only 20 years after the first fully functional computers.

NOTE

And yes, you can run computers within computers within computers with virtualization. They get slower and slower the more levels you go in, but the logic still works.

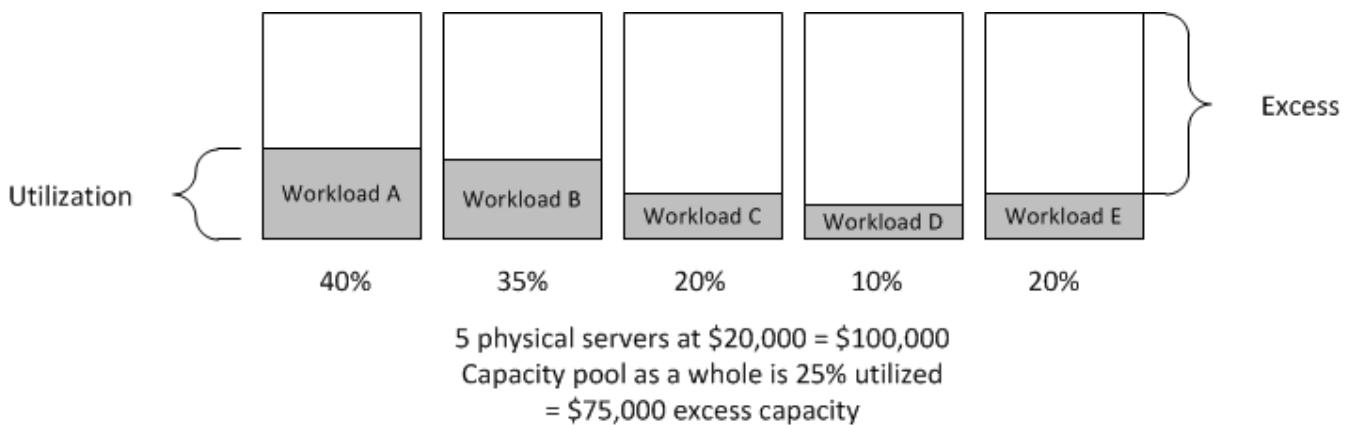
In order to run compute as a utility, multi-tenancy was essential. This is different from electricity (but similar to the phone system). As noted elsewhere, one watt of electric power is like any other and there is less concern for leakage or unexpected interactions. People's bank balances are not encoded somehow into the power generation and distribution infrastructure.

However, people store data - often private - on computers. In order to deliver compute as a utility, it is essential to segregate each customer's workload from all others. This is called multi-tenancy. In multi-tenancy, multiple customers share physical resources that provide the illusion of being dedicated.

NOTE

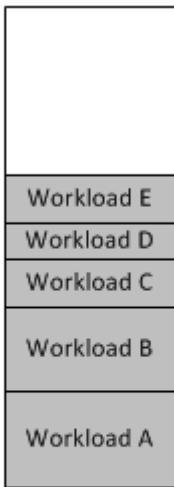
The phone system has been multi-tenant ever since they got rid of [party lines](#). A party line was a shared line where anyone on it could hear every other person.

Virtualization attracted business attention as a means to consolidate computing workloads. For years, companies would purchase servers to run applications of various sizes, and in many cases the computers were badly underutilized. Because of configuration issues (legitimate) and an overabundance of caution (questionable), average utilization in a pre-virtualization data center might average 10-20%. (That's 90% of the computer's capacity being wasted.)



The above figure is a simplification. Computing and storage infrastructure supporting each application stack in the business were sized to support each workload. For example, a payroll server might run on a different infrastructure configuration than a data warehouse server. Large enterprises needed to support hundreds of different infrastructure configurations, increasing maintenance and support costs.

The adoption of virtualization allowed businesses to compress multiple application workloads onto a smaller number of physical servers:



**1 physical server at \$40,000
with virtualized workloads
Capacity 62.5% utilized
= \$15,000 excess capacity**

NOTE

For illustration only. A utilization of 62.5% might actually be a bit too high for comfort, depending on the variability and criticality of the workloads.

In most virtualized architectures, the physical servers supporting workloads share a consistent configuration, which made it easy to add and remove resources from the environment. The virtual machines may still vary greatly in configuration, but the fact of virtualization makes managing that easier - the virtual machines can be easily copied and moved, and increasingly can be defined as a form of code (see next section).

Virtualization thus introduced a new design pattern into the enterprise where computing and storage infrastructure became commoditized building blocks supporting an ever-increasing array of services.

But what about where the application is large and virtualization is mostly overhead? Virtualization still may make sense in terms of management consistency and ease of system recovery.

NOTE

Some database professionals still don't like virtualization, that is, making them run Oracle RDBMS or Microsoft SQL Server on top of virtual servers, instead of directly on the host operating system. "The database IS virtualization" they will say.

Virtualization vs cloud

Virtualization is necessary, but not sufficient for cloud. True Cloud services are highly automated, and most Cloud analysts will insist that if virtual machines cannot be created and configured in a completely automated fashion, the service is not true Cloud. This is currently where many in-house "private" Cloud efforts struggle; they may have virtualization, but struggle to make it fully self-service.

There also is a long tradition of managed services, where applications are built out by a customer and then their management is outsourced to a third party.

NOTE

Traditional managed services are sometimes called "your mess for less." With Cloud, you have to "clean it up first."

Cloud services have refined into at least three major models:

- Infrastructure as a service
- Platform as a service
- Software as a service

From the NIST Definition of Cloud Computing (p. 2-3):

Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

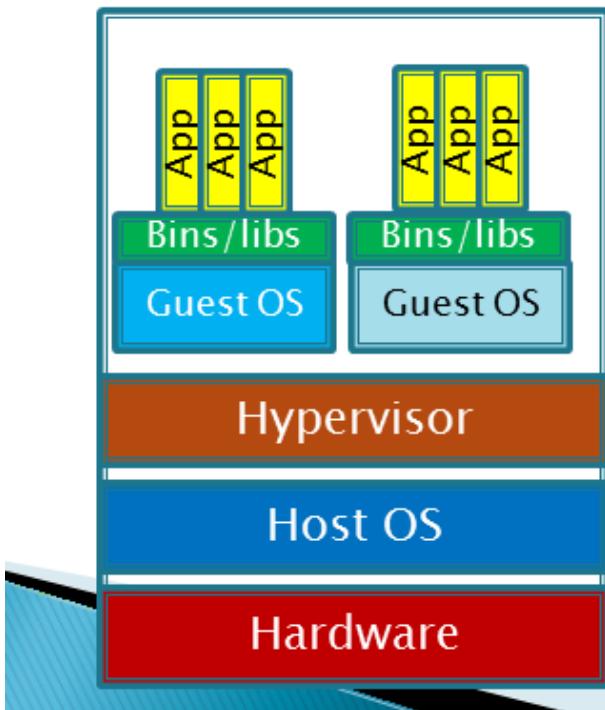
Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

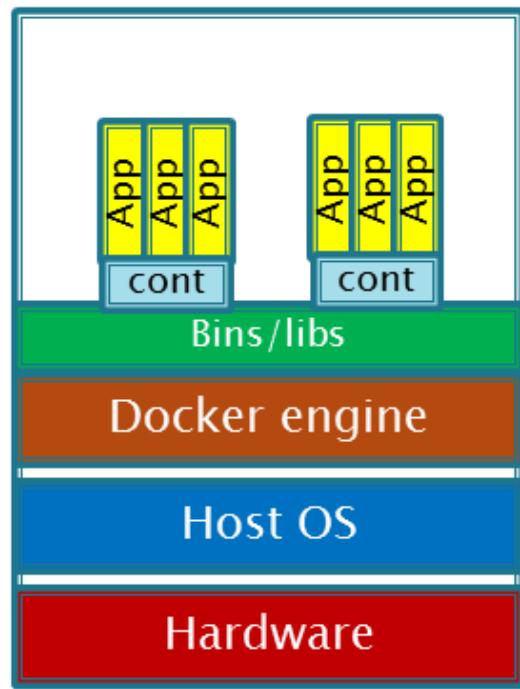
There are Cloud services beyond those listed above (e.g. Storage as a Service). Various platform services have become extensive on providers such as Amazon, which offers load balancing, development pipelines, various kinds of storage, and much more.

Containers and looking ahead

Type 2



Containers



At this writing, two major developments in Cloud computing are prominent:

- Container technologies such as [Docker](#)
 - Containers are lighter weight
 - Virtualized Guest OS: Seconds to instantiate
 - Container: Milliseconds (!)
 - Containers must be same OS as host
- [AWS Lambda](#), "a compute service that runs your code in response to events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information."

It's recommended you at least scan the links provided.

Infrastructure as code

So, what is infrastructure as code?

As cloud infrastructures have scaled, there has been an increasing need to configure many servers identically. Auto-scaling (adding more servers in response to increasing load) has become a widely used strategy as well. Both call for increased automation in the provisioning of IT infrastructure. It is simply not possible for a human being to be hands on at all times in configuring and enabling such infrastructures, so automation is called for.

In years past, infrastructure administrators relied on ad-hoc issuance of commands either at an operations console, or via a GUI-based application. Shell scripts might be used for various repetitive

processes, but administrators by tradition and culture were empowered to issue arbitrary commands to alter the state of the running system directly.

The following passage from The Phoenix Project captures some of the issues. The speaker is Wes, the infrastructure manager, who is discussing a troubleshooting scenario:

“Several months ago, we were three hours into a Sev 1 outage, and we bent over backward not to escalate to Brent. But eventually, we got to a point where we were just out of ideas, and we were starting to make things worse. So, we put Brent on the problem.” He shakes his head, recalling the memory, “He sat down at the keyboard, and it’s like he went into this trance. Ten minutes later, the problem is fixed. Everyone is happy and relieved that the system came back up. But then someone asked, ‘How did you do it?’ And I swear to God, Brent just looked back at him blankly and said, ‘I have no idea. I just did it.’”

[Kim2013], p. 116.

Obviously, “close your eyes and go into a trance” is not a repeatable process. It is not a procedure or operation that can be archived and distributed across multiple servers. So, shell scripts or more advanced forms of automation are written and increasingly, all actual server configuration is based on such pre-developed specification. It is becoming more and more rare for a systems administrator to actually “log in” to a server and execute configuration-changing commands in an ad-hoc manner (as Brent).

In fact, because virtualization is becoming so powerful, servers increasingly are destroyed and rebuilt at the first sign of any trouble. In this way, it is certain that the server’s configuration is as intended. This again is a relatively new practice.

Previously, because of the expense and complexity of bare-metal servers, and the cost of having them offline, great pains were taken to fix troubled servers. Systems administrators would spend hours or days troubleshooting obscure configuration problems, such as residual settings left by removed software. Certain servers might start to develop “personalities.” Industry practice has changed dramatically here since around 2010.

A simple infrastructure as code example

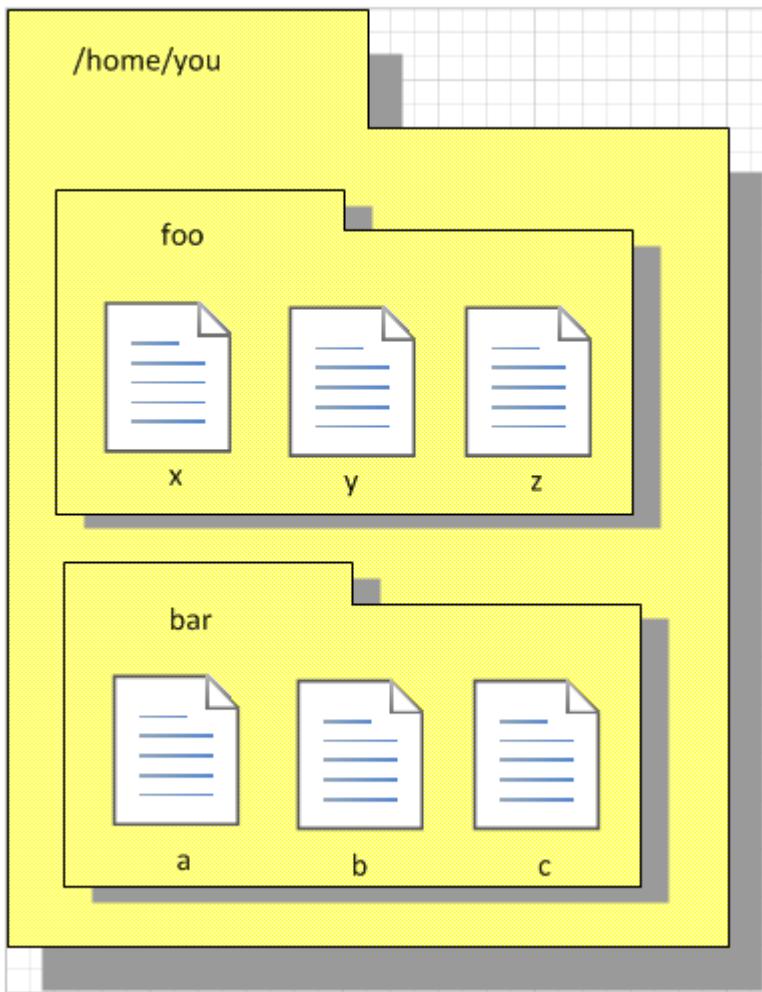
Note, the below section is illustrative only, and is not intended as a lab. The associated lab for this book goes into much more depth on these topics.

In presenting infrastructure as code at its simplest, we will start with the concept of a shell script. While this is not a deep Linux book (there are many others out there, starting with the excellent O’Reilly lineup), some basic technical literacy is assumed in this book. Consider the following set of commands:

```
$ mkdir foo  
$ cd foo  
$ touch x y z  
$ cd ..  
$ mkdir bar  
$ cd bar  
$ touch a b c
```

What does this do? It tells the computer, “Create two directories, one named foo and one named bar. In the one named foo, create three blank files named x,y, and z; in the one named bar, create three blank files, named a, b, and c.”

If you find yourself (with the appropriate permissions) at a Unix command prompt, and run those commands, you will wind up with a configuration that could be visualized as this:



(If you don't understand this, you should probably spend a couple hours with a Linux tutorial).

Configuration, you ask? Something this trivial? Yes, directory and file layouts count as configuration and in some cases are critical.

Now, what if we take that same set of commands, and put them in a file thus:

```
#!/bin/bash
mkdir foo
cd foo
touch x y z
cd ..
mkdir bar
cd bar
touch a b c
```

We might name that file iac.sh, set its permissions correctly, and run it (so that the computer executes all the commands for us, rather than us running them one at a time at the console). If we did so in an empty directory, we'd again wind up with that same configuration. (If we did it in a directory already containing foo and bar directories, we'd get errors. More on this to come.)

This will be familiar material to many of you, including the fact that beyond creating directories and files we can use shell scripts to create and destroy virtual servers, install and remove software, set up and delete users, check on the status of running processes, and much more.

The state of the art in infrastructure configuration is not to use shell scripts at all but rather policy-based infrastructure management approaches ...

Let's return to our iac.sh file. It's valuable. It documents our intentions for how this configuration should look. We can reliably run it on thousands of machines and it will always give us two directories and six files. In terms of the previous section, we might choose to run it on every new server we create. We want to establish it as a known resource in our technical ecosystem. This is where version control comes in.

Version control: the Agile foundation

In software development, version control is the foundation of every other Agile technical practice. Without version control, there is no build, no TDD, no continuous integration.

— Andrew Clay Shafer, in Allspaw-Web Operations: Keeping the Data On Time

cite Forstgren <http://www.slideshare.net/nicolefv/nf-final-agileindia2016> - evidence on centrality of version control

What is version control?

Now, because this is a general book on IT management, we're going to take a bit of a different turn here. As mentioned previously, there are numerous sources available to you to learn Linux scripting, policy-based configuration management of infrastructure, and source control. Competency with source control is essential to your career and you should devote some serious time to it. This book has a different purpose: to orient you to the fundamental principles and

architectures of IT management at all scales.

We are going to take a very deep and targeted look at version control, in particular source control, because it is here that we start to see the emergence of an architecture of IT management. It is in the source control system that we first start to see metadata emerge as an independent concern.

(For a definition of metadata, see the sidebar.)

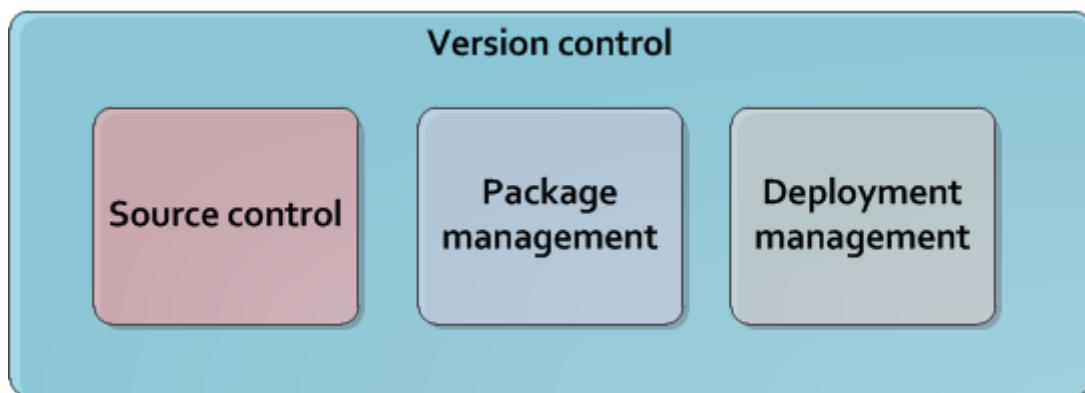
The concept of a software version control system (variously known as source control, revision control, software configuration management, and other terms) is our first major software system for the “business of IT.” (The virtualization architecture previously discussed is what is under management; the source control system is part of the management architecture.)

The evolution of version control systems to the pivotal role they now hold has been somewhat unexpected. While version control was always deemed important for software artifacts, it has only recently become the preferred paradigm for managing infrastructure state as well. Because of this, a version control system is easily and without question the first IT management system you should acquire and implement.

It's significant that the Agile Alliance indicates “version control” as one of the [four foundational areas of Agile](#), along with team, iterative development, and incremental development.

NOTE

“Version control” also includes what we will cover as package management: the management of binary files, as distinct from human-understandable symbolic files. There is also a need to understand what versions are installed on what computers; this can be termed “deployment management.” (With the advent of containers, this is a particularly fast changing area.)



Source control works like an advanced file system with a memory. (Actual file systems that do this are called [versioning](#) file systems.) It can remember all the changes you make to its contents, tell you the differences between any two versions, and also bring back the version you had at any point in time.

Source control, as you may imagine, is critical for any kind of system with complex, changing content, especially when many people are working on that content, and there is a need to see the exact sequence of its evolution and isolate any particular moment in its history, or provide detailed analysis on how two versions differ.

You can find many references to source control on the Internet and in books such as *Pro Git* by Scott Chacon and Ben Straub [[Chacon2009](#)]. As it is the most important foundational technology for professional IT, whether in a garage or in the largest corporations, you need to have a deep familiarity with it.

Source control is at its most powerful when dealing with symbolic data, which we usually see as text files. It is less useful in dealing with unstructured, binary data, such as image files.

This is because symbolic files can be “differenced” in a way that is meaningful to humans. If I change “abc” to “abd” it is clear that the third character has been changed from “c” to “d.”

On the other hand, if I take a picture (e.g. as a JPEG file) and alter one pixel, and compare the resulting before and after binary files in terms of their data, it would be more difficult to understand what had changed. I would be able to easily tell that they are two different files (they would have different checksums), but they would look very similar.

consider an illustration

IT starts with symbolic files. Text editors create source code, scripts, and configuration files. These may be transformed in defined ways (e.g. by compilers and build tools) but the human understandable end of the process is mostly based on text files.

We care very much about when a text file changes. One wrong character can completely alter the behavior of a large, complex system. Therefore, our configuration management approach must track to that level of detail.

Although implementation details may differ, all version control systems have some concept of “commit.” As stated in *Version Control with Git* [[Loeliger2009](#)]:

In Git, a commit is used to record changes to a repository . . . Every Git commit represents a single, atomic changeset with respect to the previous state. Regardless of the number of directories, files, lines, or bytes that change with a commit... either all changes apply or none do. In terms of the underlying object model, atomicity just makes sense: A commit snapshot represents the total set of modified files and directories... Git doesn't care why files are changing. That is, the content of the changes doesn't matter. As the developer, you might move a function from here to there and expect this to be handled as one unitary move. But you could, alternatively, commit the removal and then later commit the addition. Git doesn't care. It has nothing to do with the semantics of what is in the files. [emphasis added]

The concept of a version or source control “commit” is a rich foundation for IT management and governance. It both represents the state of the computing system as well as providing evidence of the human activity affecting it.

As we will see in Chapter 3, the “commit” identifier is directly referenced by build activity, which in turn is referenced by the release activity, which is typically visible across the IT value chain.

Also, the concept of an atomic “commit” is essential to the concept of a “branch” - the creation of an experimental version, completely separate from the main version, so that various alterations can be tried without compromising the overall system stability. Starting at the point of a “commit,” the

branched version also becomes evidence of human activity around a potential future for the system. In some environments, the branch is automatically created with the assignment of a requirement or story - again, more on this to come in chapter 3. In other environments, the very concept of branching is avoided.

Discussion of branching & merging?
source control versus package management

Sidebar: What is metadata?

An understanding of the term "metadata" is required for this book's approach to IT management.

Metadata is a tricky term, that tends to generate confusion. The term “meta” implies a concept that is somehow self-referential, and/or operating at a higher level of abstraction. So,

- the term meta-discussion is a discussion about the discussion;
- meta-cognition is cognition about cognition, and
- meta-data (aka metadata) is data about data.

Some examples:

- In traditional data management, metadata is the description of the data structures, especially from a business point of view. A database column might be named “CUST_L_NM,” but the business description or metadata would be “The last, family, or surname of the customer.”
- In document management, the document metadata is the record of who created the document and when, when it was last updated, and so forth. Failure to properly sanitize document metadata has led to various privacy and data security related issues.
- In telephony, “data” is the actual call signal — the audio of the phone conversation, nowadays usually digitally encoded. Metadata on the other hand is all the information about the call: from who to who, when, how long, and so forth.

In computer systems, metadata can be difficult to isolate. Sometimes, computing professionals will speak of a “metadata” layer that may define physical database structures, data extracts, business process behavior, even file locations. The trouble is, from a computer’s point of view, a processing instruction is an instruction, and the prefix “meta” has no real meaning.

Because of this, this book favors a principle that **metadata is by definition non-runtime**. It is documentation, usually represented as structured or semi-structured data, but not usually a primary processing input or output. It might be “digital exhaust” - log files are a form of metadata. It is not executable. If it’s executable (directly or indirectly), it’s digital logic or configuration, plain and simple.

So what about our infrastructure as code example? The artifact - the configuration file, the script - is NOT metadata, because it is executable. But the source repository commit IS metadata. It has no meaning for the script. The dependency is one way - without the artifact, the commit ID is

meaningless, but the artifact is completely ignorant of the commit. However, the commit may become an essential data point for human beings trying to make sense of the state of a resource defined by that artifact.

In this microcosm, we see the origins of IT management. It is not always easy to apply this approach in practice. There can be edge cases. But **the concept of metadata provides a basis for distinguishing the *management* of information technology from the actual *practice* of information technology.**

Policy-based approaches

DRAFT [As of fall 2015 this section needs to be elaborated.]

Before we turned to source control, we looked at a simple script that changed the configuration of a computer. It did so in an *imperative* fashion.

NOTE

Imperative vs Declarative. These are two important terms from computer science. In an imperative approach, we tell the computer specifically how we want to accomplish a task. Many traditional programming languages take an imperative approach. Whereas with a declarative approach, we tell the computer the task we want to accomplish and let the computer figure out the best way to do it. Structured Query Language (SQL) statements are a great example of a declarative approach. We can provide a database system with a set of SQL statements, and the database will generate an execution plan to provide us with the data we are seeking.

A script such as iac.sh is executed line by line, i.e., it is imperative. But in policy-based approaches, the object is to define the desired end state of the resource, not the steps needed to get there.

Chef, Puppet, Salt Stack, and Ansible are all declarative to some degree.

This is a complex topic and there are advantages and disadvantages to each approach. (See [When and Where Order Matters](#) by Mark Burgess for an advanced discussion.) But policy-based approaches seem to have the upper hand for now.

describe modern infra managers, policy management, auto-scaling as an example, provide an example Chef script..

Topics in IT infrastructure

===== Web-scale IT

This chapter as of fall 2015 is yet to be written. Will summarize key insights from the site reliability community, including Allspaw and Limoncelli.

===== CAP theorem
to be written

===== Technology lifecycle management

[to be written]

===== additional material

<http://12factor.net/build-release-run>

Releases are an append-only ledger and a release cannot be mutated once it is created.
Any change must create a new release.

In the old world, environments were accepted to be different except where they needed to be consistent. In the new world, environments are assumed to be the same except where they need to be different.

what makes a good infrastructure engineer

<http://jpmorgenthal.com/2016/08/24/a-reality-check-on-everyones-moving-everything-to-the-cloud/> excellent

Conclusion

Books and articles are written every week about some aspect of IT infrastructure. We have only scratched the surface in our discussions of compute, network, and storage, and how they have become utility services in the guise of Cloud.

Software as a service, platform as a service, infrastructure as a service - each represents a different approach. For the most part, we will focus on Infrastructure as a Service in the remainder of this book, on the assumption that your digital product is unique enough to need the broad freedom this provides.

IT infrastructure is a rich and complex topic, and a person can spend their career specializing in it. For this class, we always want to keep chapter 1 in mind: why do we want it? How does it provide for our needs, contribute to our enjoyment?

Discussion questions

1. Consider your product idea. Does it have any particular infrastructure needs you can identify, having read this chapter?

Research & practice

1. Research Cloud providers and recommend which you would prefer to build your new product on.

Further reading

Books

- Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan, [The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Volume 2](#)
- John Allspaw, [Web Operations: Keeping the Data On Time](#)
- Ellen Siever, Stephen Figgins, Robert Love (Goodreads Author), Arnold Robbins, Aaron Weber, [Linux in a Nutshell](#)
- Scott Chacon, [Pro Git](#)

Articles

- [Are your servers pets or cattle?](#)
- [Clouds Jumping – Why AWS Is Kicking PaaS](#)
- [Latency Numbers Every Programmer Should Know](#)
- [NIST definition of Cloud computing](#)
- [Google Is 2 Billion Lines of Code—And It's All in One Place](#)
- [On DVCS, continuous integration, and feature branches](#)

Picture credits

<https://www.flickr.com/photos/arbron/56216585/>

2nd draft
risk management & infrastructure choices (Charity Majors talk 7/21/2016 DevOpsDays MSP)

Chapter 3: Application Delivery

Instructor's note I have opted to defer the "theory" of Agile (as defined by Reinertsen) to Chapter 4. So, this chapter presents Agile and related concepts like iterative development without examining the underlying principles.

I do this because I have discovered that theory sometimes works better in retrospect. Many students increasingly come in with some exposure to Cloud and Agile methods at least, and Chapters 2 and 3 will seem comfortable and familiar. In Chapter 4 we challenge them with **why** Agile works.

Introduction to Chapter 3

Now that we have some idea of IT value (and how we might turn it into a product), and have decided on some infrastructure, we can start building.

IT systems that directly create value for non-technical users are usually called "applications," or sometimes "services." As discussed in chapter 1, they enable value experiences in areas as diverse as consumer banking, entertainment and hospitality, and personal transportation. In fact it is difficult to think of any aspect of modern life untouched by applications. (This overall trend is sometimes called digital transformation.)

Applications are built from software, the development of which is a core concern for any IT-centric product strategy. Software development is a well established career, and a fast-moving field with new technologies, frameworks, and schools of thought emerging weekly, it seems.

This chapter will cover applications and the software lifecycle, from requirements through construction, testing, building, and deployment into modern production environments. It also discusses earlier approaches to software development, the rise of the Agile movement, and its current manifestation in the practice of DevOps.

Chapter outline

- What is an "application"
- How applications have been delivered over time
- Application lifecycle
- Introduction to Agile
- Introduction to DevOps
 - Application toolchain
- Test-driven development
- Continuous integration
- Continuous deployment

Learning objectives

- Understand history and importance of "application" concept

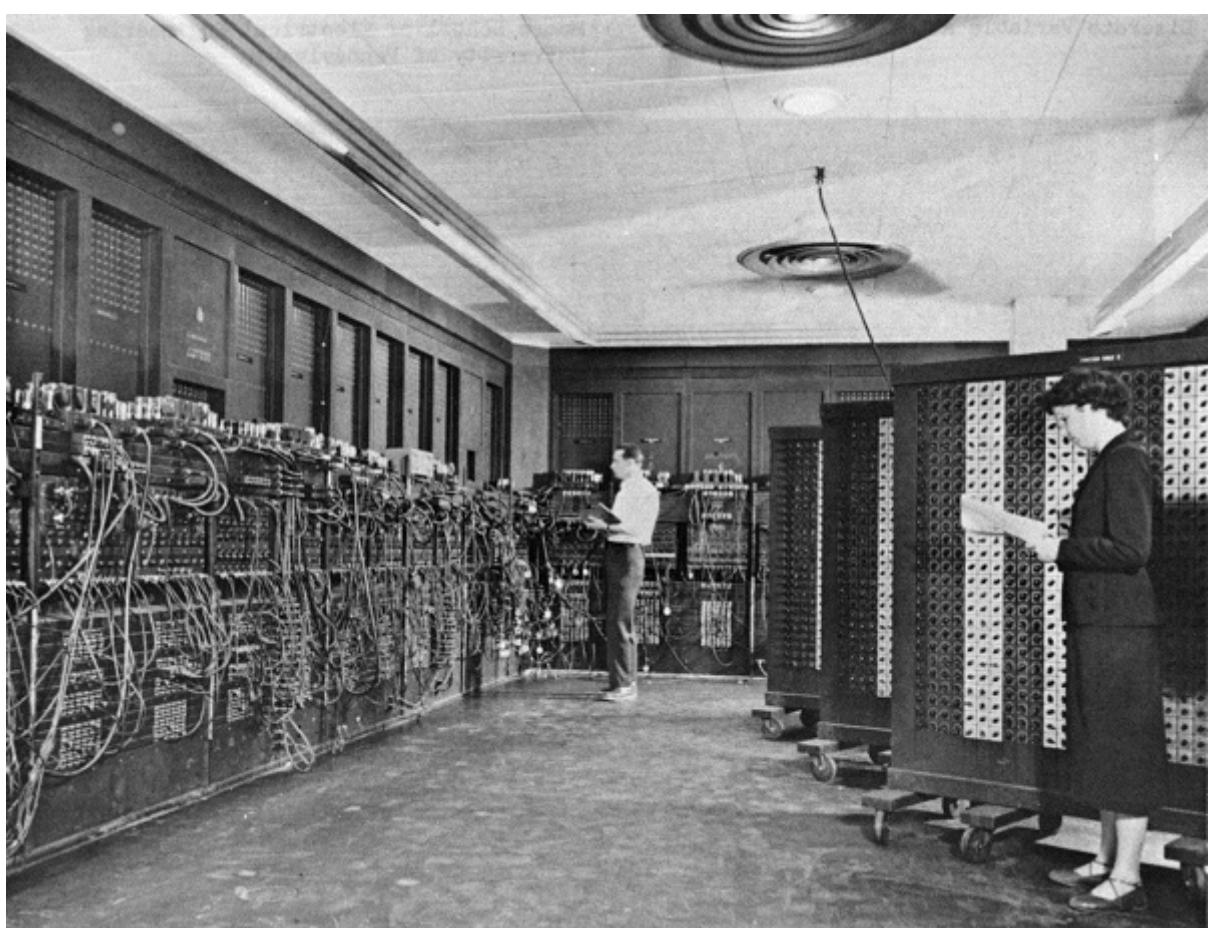
- Define "Agile" in terms of software development
- Identify key Agile practices
- Identify the major components of an end-to-end DevOps delivery pipeline.

Basics of applications and their development

Defining "Application"

In keeping with our commitment to theory and first principles, we use an engineering definition of "application." To an electrical engineer, a toaster or a light bulb is an "application" of electricity. Similarly, a customer relationship management system, or a Web video on demand service, are "applications" of the core computer infrastructure we studied in the last chapter.

History of applications and application software



Public domain photo of early ENIAC computer from Wikipedia. ENIAC was "programmed" by reconfiguring its cables, not by a programming language.

Without applications, computers would be merely a curiosity. Computers were first "applied" to military needs for codebreaking and artillery calculations. After World War II, ex-military officers like [Edmund Berkeley](#) at Prudential realized their potential if "applied" to problems like insurance record keeping. [\[Akera2007\]](#)

At first, such systems required actual manual configuration (as in the ENIAC photo in this section), or painstaking programming in complex, tedious, and unforgiving low-level [programming](#)

[languages](#). As the value of computers became obvious, investment was made in making programming easier through more powerful languages.

The [history of software](#) is well documented. Low level languages ([binary](#) and [assembler](#)) were increasingly replaced by higher level languages ([FORTAN](#), [COBOL](#), [C](#)). Proprietary machine/language combinations were replaced by open standards and [compilers](#) that could take a common set of source code and build it for different hardware platforms.

Today, we have extensive frameworks like Struts and Ruby on Rails, along with interpreted languages that take much of the friction out of building and testing code. But even today, the objective remains to create a [binary executable](#) file or files that computer hardware can "execute," that is, turn into a computing-based value experience, mediated through devices such as workstations, laptops, smart phones, and their constituent components.

Applications and infrastructure: the old way

In the first decades of computing, any significant application of computing power to a new problem typically required its own [infrastructure](#), often designed specifically for the problem. While computer scientists always were aware that computers in theory could be "general purpose," in practice this was not so easy. Military/aerospace needs differed from industry, which differed from scientific and technical uses. And major new applications required new compute capacity.

Take for example when a corporation in 1998 decided to replace its mainframe Human Resources system due to [Y2K concerns](#). Such a system might need to support several thousand users around the world. At that time, PeopleSoft was a frequent choice of software. Implementing such a system was often led by consulting firms such as Deloitte or Andersen Consulting (where the author worked).

A typical PeopleSoft package implementation would include:

- [PeopleSoft](#) software, including the PeopleTools framework and various modules written in the framework (e.g. the well-regarded PeopleSoft HR system)
- [Oracle database](#) software
- AT&T "Tuxedo" [transaction manager](#)
- Autosys [job scheduler](#)
- [HP-UX operating system](#)
- HP-UX servers, perhaps 20 or so, comprising various "environments" including a production "cluster" consisting of application and database servers
- [EMC storage array](#)
- Various ancillary software and hardware: management utilities and scripts, backup, networking, etc.
- Customization of the PeopleSoft HR module and reports by hired consultants, to meet the requirements of the acquiring organization

The software and hardware needed to be specified in keeping with requirements, and acquiring it took lengthy negotiations and logistics and installation processes. Such a project from inception to

production might take 9 months (on the short side) to 18 or more months.

Hardware was dedicated and rarely re-used. The HP servers compatible with PeopleSoft might have few other applications, if they became surplus. In fact, PeopleSoft would "certify" the infrastructure for compatibility. Upgrading the software might require also upgrading the hardware.

In essence, this was more akin to [systems engineering](#), as designing and optimizing the hardware component was a significant portion of the work.

Applications and infrastructure today

Today, matters are quite different, and yet echoes of the older model persist. As mentioned, ANY compute workloads are going to incur economic cost. However, capacity is being used more efficiently and can be provisioned on demand. Currently, it is a significant application indeed that merits its own systems engineering.

Instead, a variety of mechanisms (as covered in the previous chapter's [discussion of Cloud systems](#)) enable the sharing of compute capacity, the raw material of application development. The fungibility and agility of these mechanisms increase the velocity of creation and evolution of application software.

NOTE

To "provision" in an IT sense means to make the needed resources or services available for a particular purpose or consumer.

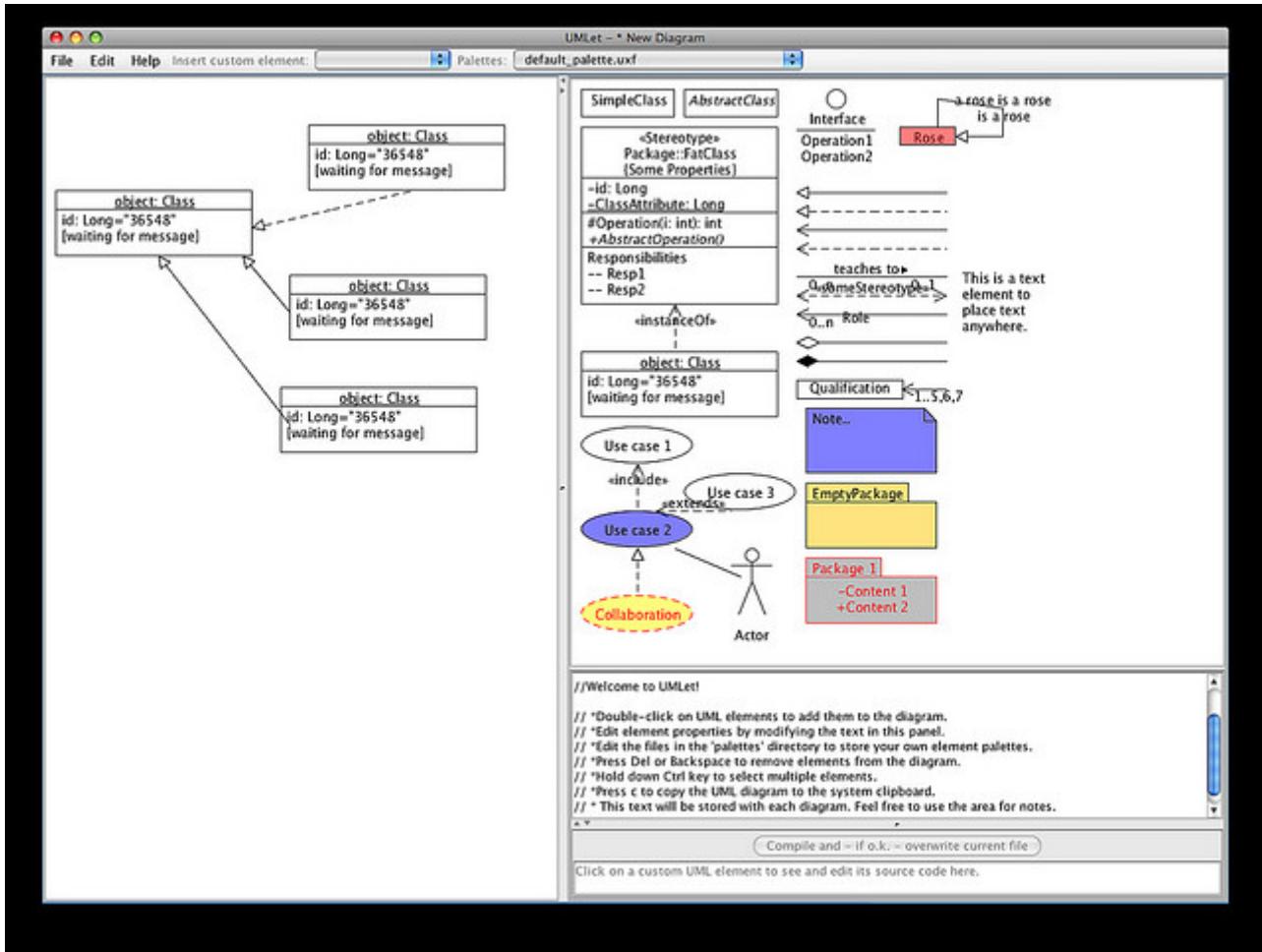
For small and medium sized applications, the overwhelming trend is to virtualize and run on commodity hardware and operating systems. Even 15 years ago, non-trivial web sites with database integration would be hosted by internal [PaaS](#) clusters at major enterprises (for example, Microsoft ASP, COM+, and SQL Server clusters could be managed as multi-tenant).

The general purpose capabilities of virtualized public and private Cloud today are robust. Assuming the organization has the financial capability to purchase computing capacity in anticipation of use, it can be instantly available when the need surfaces.

Interfaces (interaction points for applications to exchange information with each other, generally in an automated way) are increasingly standardized. Hardware engineering is more and more independent of the application lifecycle; the trend is towards providing compute as a service, carefully specified in terms of performance, but NOT particular hardware.

Hardware physically dedicated to a single application is rarer, and even the largest engineered systems are more standardized so that they may one day benefit from Cloud approaches.

Application architecture



Software architecture design tool

Photo credit: <https://www.flickr.com/photos/osde-info/2786012027>, Creative commons license

The design and architecture of applications is a large topic (see e.g. [\[Gamma1995\]](#) and [\[Fowler2003\]](#)) and this text only touches lightly on it. [ISO/IEC 41010](#) defines architecture as “the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”

A computer program can be as simple as [Hello World](#). Such a program requires only one or a few files to compile and execute. However, significant applications such as an online video streaming service require hardware and software configurations of [tremendous complexity](#). Specialized visual notations are used to describe this complexity, as in the image above. We will discuss this further in Chapter 11.

Applications and project management

Because the initial applications were implemented as a kind of systems engineering, and were expensive to build and maintain, the technique of choice was [project management](#). Project management will not be covered in this book until Chapter 8, as it is not appropriate to the earlier stages of this book’s [emergence model](#).

This history of project managed systems engineering produced any number of successes, but by the early 1990s there were significant concerns with the rate of large project failures [\[Glass1998\]](#), which occurred despite seemingly extensive and rigorous bureaucratic overhead, evidenced by

frameworks such as [CMMI](#) and [PMBOK](#). [7: Both project management and CMMI have come in for significant criticism [\[Koskela2002\]](#), [\[Keefer2006\]](#), and will be discussed further in Sections 3 and 4.]

Packaged software acquisition and integration

[to be written]

History of the Agile movement

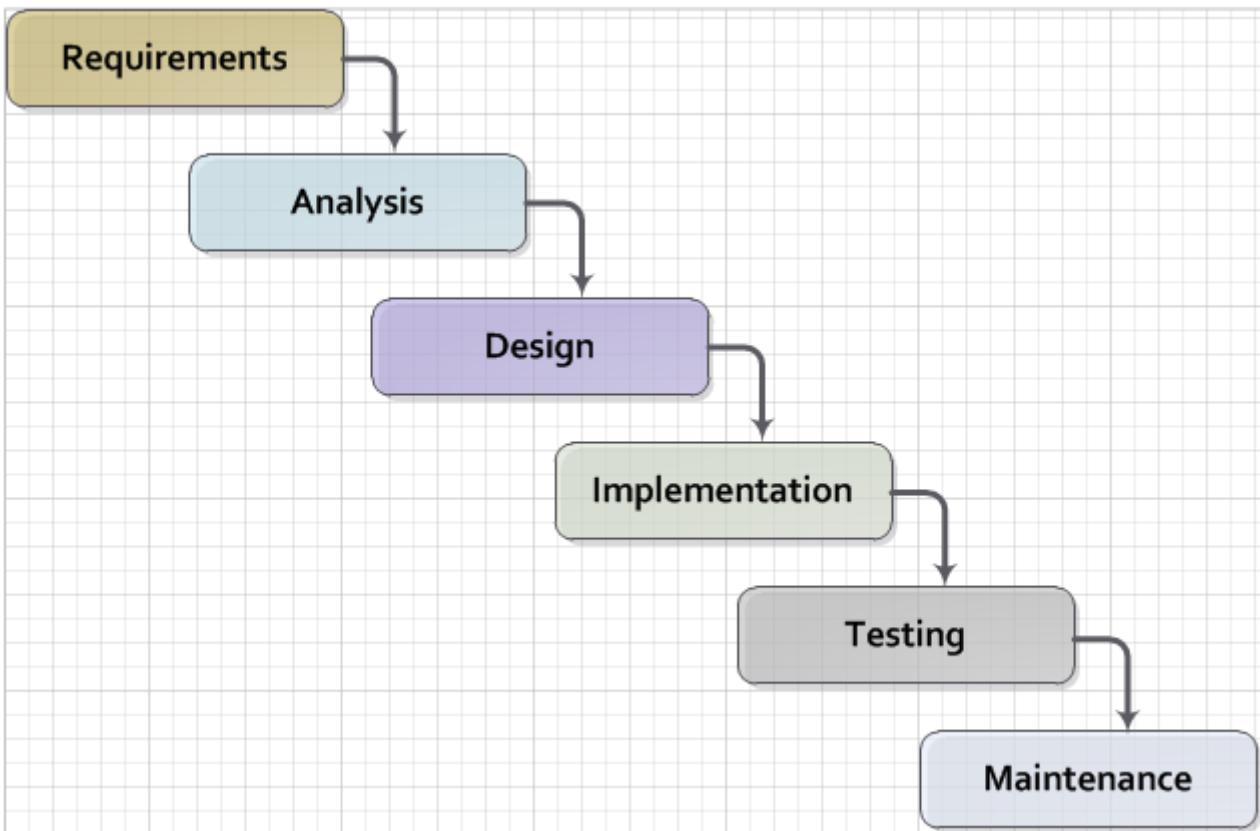
This is not a book on software development per se, nor on Agile development. There are hundreds of books available on those topics. But, no assumption is made that the reader has any familiarity with these topics, so some basic history is called for. (If you have taken an introductory course in software engineering, this will likely be review.)

As we will cover in more detail in the next chapter, the product development process starts with a concept of requirements (whether we call it story, use case, or scenario is not important). Requirements are numerous and evolving, and we're going to take some time looking at the process of converting them into IT-based functionality. There is history here back to the earliest days of computing.

When the author first joined Andersen Consulting (now Accenture) in 1998, we were schooled in something called the Business Integration Method, or BIM. The BIM was a classic expression of what is called "waterfall development".

What is waterfall development? It is a controversial question. The original theorist who coined the term named it in order to critique it [\[Royce1970\]](#). Military contracting and management consultancy practices, however, embraced it, as it provided an illusion of certainty. The fact that computer systems until recently included a substantial component of hardware systems engineering may also have contributed.

Waterfall development as a term has become associated with a number of practices. The original illustration was similar to this:



First, requirements need to be extensively captured and analyzed before the work of development should commence.

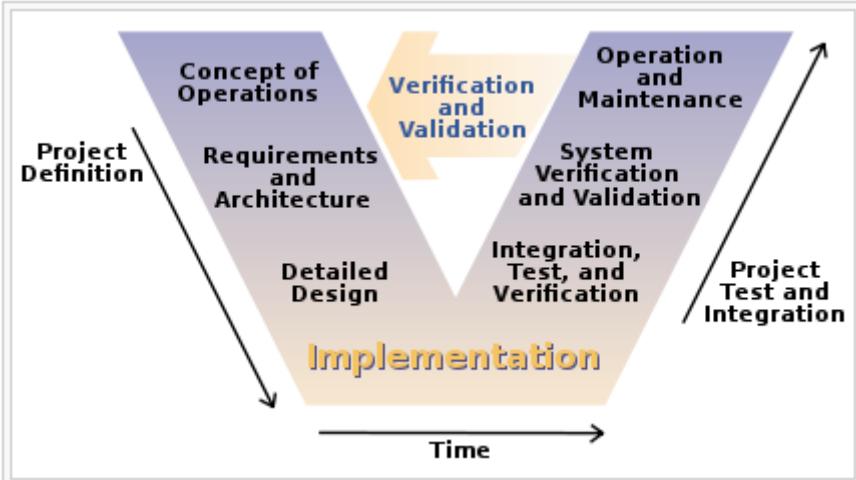
So, we would develop enormous spreadsheets of requirements, spending weeks on making sure that they represented what “the customer” wanted. The objective was to get the customer’s signature. Any further alterations could be profitably billed as “change requests”.

The analysis phase was used to develop a more structured understanding of the requirements, e.g. conceptual and logical data models, process models, business rules, and so forth.

In the design phase, the actual technical platforms would be chosen; major subsystems determined with their connection points, initial volumetrics translated into system sizing, and so forth. (Perhaps hardware would not be ordered until this point, leading to issues with developers now being “ready”, but hardware not being available for weeks or months yet).

Only AFTER extensive requirements, analysis, and design would coding take place (implementation). Furthermore, there were separation of duties between developers and testers. Developers would write code and testers would try to break it, filing bug reports that the developers would then need to respond to.

Another model sometimes encountered at this time was the V-model. This was intended to better represent the various levels of abstraction operating in the systems delivery activity. Requirements operate at various levels, from high level business intent through detailed specifications. It is all too possible that a system is “successfully” implemented at lower levels of specification, but fails to satisfy the original higher level intent.



Public domain image of v-model from [Wikipedia](#)

The failures of these approaches at scale are by now well known. Large distributed teams would wrestle with thousands of requirements. The customer would “sign off” on multiple large binders, with widely varying degrees of understanding of what they were agreeing to. Documentation became an end in itself, and yet did not meet its objectives of ensuring continuity if staff turned over.

The development team would design and build out extensive product implementations without checking the results with customers. They would also defer testing that various component parts would effectively interoperate until the very end of the project, when the time came to assemble the whole system.

Failure after failure of this approach is apparent in the historical record ([Glass1998]). This led to the perception of a “software crisis.” (It should also be noted that many large systems were effectively constructed and operated during the “waterfall years”, and that there are reasonable criticisms of the concept of a “software crisis” [Bossavit2015]).

Successful development efforts existed back to the earliest days of computing (otherwise, we probably wouldn’t have computers, or at least not so many). Many of these successful efforts used prototypes and other means of building understanding and proving out approaches. But highly publicized failures continued, and a substantial movement against “waterfall” development started to take shape.

By the 1990s, a number of thought leaders in software development had noticed some common themes with what seemed to work and what didn’t. Perhaps the first and best known was Kent Beck, who developed a methodology known as “eXtreme Programming,” or XP [Beck2000]. XP pioneered the concepts of iterative, fast-cycle development with ongoing stakeholder feedback, coupled with test-driven development, ongoing refactoring, pair programming, and other practices. (More on the specifics of these in the next section.)

Various authors assembled in 2001 and developed the Agile Manifesto [Alliance2001], which further emphasized an emergent set of values and practices:

The Agile Manifesto We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

The Manifesto authors further stated:

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

See <http://agilemanifesto.org/>.

The Agile models for developing software aligned with the rise of Cloud and Web-scale IT. As new customer-facing sites like Flickr, Amazon, Netflix, Etsy, and Facebook scaled to massive proportions, it became increasingly clear that waterfall approaches were incompatible with their needs. The sheer size and complexity of these systems required much more incremental and iterative approaches to delivery.

Furthermore, because these systems were directly user-facing, delivering monetized value, they required a degree of responsiveness previously not seen in “back-office” IT or military-aerospace domains (the major forms that large scale system development had taken to date).

Web-based systems integrate the software development lifecycle tightly with operational concerns. The development of new functionality is moved rapidly into a user-facing state, as opposed to previous models where software development was more distant in time and personnel from operations staff. We will talk more of product-centricity and the overall DevOps movement in the next section.

Software was moving more directly into an operational state, and developers and operators were part of the same economic concern (contract software development never gained favor in the Silicon Valley web-scale community). So, it was possible to start breaking down the walls between “development” and “operations,” and that is just what happened.

This new world also did not function at all in terms of large requirements specifications. Capturing a requirement, analyzing and designing to it, implementing it, testing that implementation, and deploying the result to the end user for feedback became something that needed to happen at speed, with high repeatability. Requirements “backlogs” were (and are) never “done,” and increasingly were the subject of ongoing re-prioritization, without high-overhead project “change” barriers.

These large scale web properties also started to “test in production” (more on this in Section 2) in the sense that they would deploy new functionality to only some of their users. Because large scale systems are complex and unpredictable, it is understood that new features are never fully understood until they are deployed at scale to the real end user base.

Rather than trying to increase testing to better understand things before deployment, these new firms accepted a seemingly higher level of risk in exposing new functionality sooner. (Part of their belief is that it actually is not higher risk, because the impacts are never fully understood in any event.) This has paid off in many cases.

When waterfall may be appropriate

[Packaged software acquisition - relatively more sequential - but open-loop approaches are always more risky - open loop not defined yet -]

- Pairing, swarming, mobbing <http://www.jrothman.com/mpd/project-management/2014/07/how-pairing-swarming-work-why-they-will-improve-your-products/>

The DevOps challenge

Add in Puppet Labs State of DevOps insights

Consider this inquiry by Mary and Tom Poppendieck ([\[Poppendieck2007\]](#), p. 92):

How long would it take your organization to deploy a change that involved one single line of code? Do you deploy changes at this pace on a repeat, reliable basis?

The implicit goal is that the organization should be able to change one line of code, and in fact might want to do so on an ongoing basis.

There is deep Lean/Agile theory behind this goal, theory developed in reaction to the pattern of massive software failures that characterized IT in the first fifty years of its existence. (We'll discuss some of the systems theory, including the concept of feedback, in the next chapter.)

Achieving this goal is feasible but requires new approaches. Various practitioners have explored this problem, with great success. Key initial milestones included:

- The establishment of “test-driven development” as a key best practice in creating software [[Beck2000](#)]
- Duvall’s book “Continuous Integration” [[Duvall2007](#)]
- Allspaw & Hammonds’s seminal “[10 Deploys a Day](#)” presentation describing Etsy [[Allspaw2009](#)].
- Humble & Farley’s “Continuous Delivery” [[Humble2011](#)].

While definitions of "DevOps" are controversial, the fundamental goal of moving smaller changes more quickly through the pipeline is common to all. There is a great deal written on the topic of DevOps currently; the Humble/Farley book is recommended as an introduction.

Let's go into a little detail on some essential Agile/DevOps practices.

- Test driven development
- Ongoing refactoring
- Continuous integration
- Continuous deployment

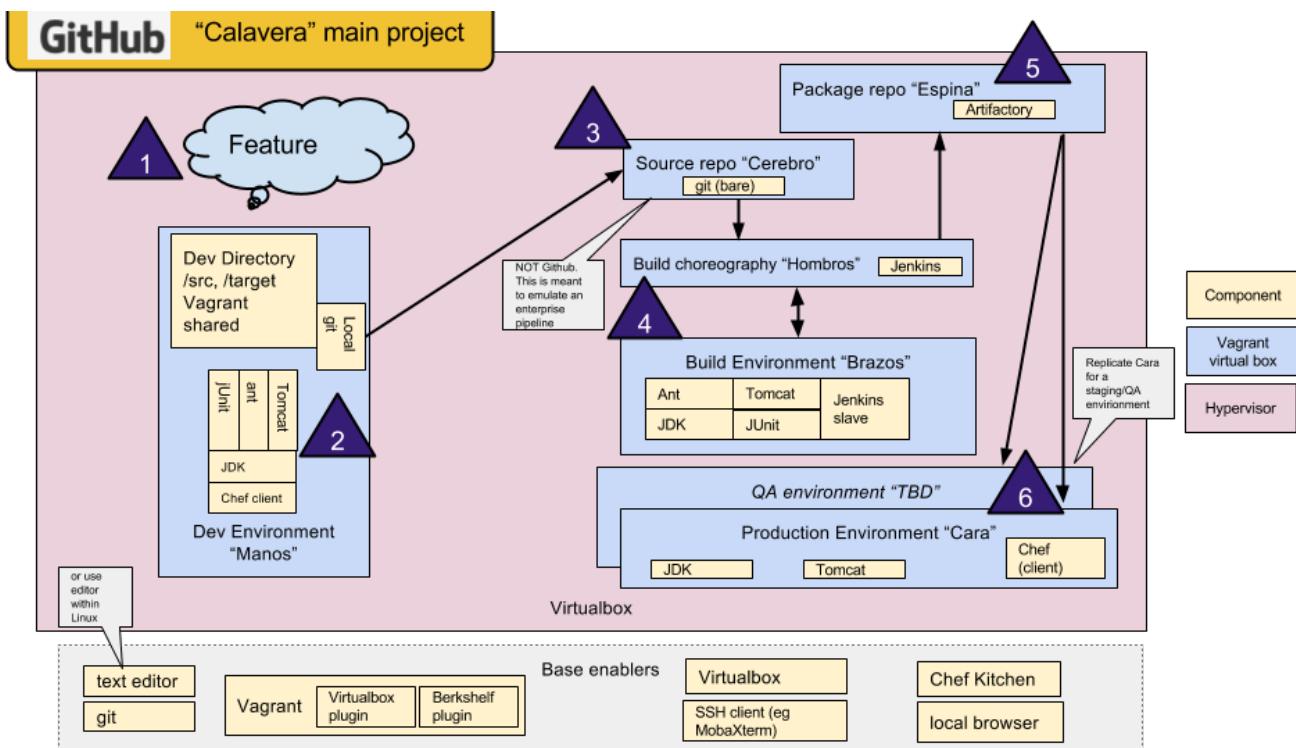
In our scenario approach, at the end of the last chapter, you had determined a toolset for creating your new IT-based product, and a platform for exposing it to users (if only yourself or your partner).

You'll be creating text files of some sort, and almost certainly importing various additional libraries, packages, modules, etc, rather than solving problems others have already figured out.

NOTE Development tools such as text editors and integrated development environments (IDEs) are out of scope for this book, as they are often matters of personal choice and limited to developers' desktops.

The assumption in this chapter is that you are going to start IMMEDIATELY with a continuous delivery pipeline. **You want to set this up before developing a single line of code.** It's not that difficult (see the online resources for further discussion and pointers to relevant open source projects).

What is meant by a continuous delivery pipeline? Here is a simplified, starting overview, based on the [Calavera project](#) (the basis for this course's labs):



1. First, some potential for value is identified. It is refined through product management techniques into a feature - some specific set of functionality that when complete will enable the value proposition (i.e. as a [moment of truth](#).)
2. The feature is expressed as some set of IT work, today usually in small increments lasting between 1 and 4 weeks (this of course varies). Software development commences, e.g. the creation of Java components by developers who first write tests, and then write code that satisfies the test.
3. The developer is continually testing the software as the build progresses, and keeping a local source control repository up to date with their changes at all times. When she or he reaches a level of satisfaction with the software, it is submitted to a centralized source repository.
4. When the repository detects the new "check-in," it launches a dedicated environment to build and test the new code. The environment is likely configured using [infrastructure as code](#) techniques; in this way, it can be created automatically and quickly.
5. If the code passes all tests, the compiled and built binary executables may then be "checked in" to a package management repository.
6. From the package repository, the code may then be deployed to various environments, for further testing and ultimately to "production," where it can enable the consumer's value experiences.

Let's go into more detail on the major phases.

Describing system intent

So, you've got an idea for a product value experience, and you have tools for creating it and infrastructure for running it. It's time to start building shippable product.

In order to do this, you need to express what you need the product to do. The conceptual tool used to do this is called Requirement. The literal word "Requirement" has fallen out of favor with the rise of Agile [\[Patton2014\]](#), and has a number of synonyms and variations:

- Use case
- User story
- Nonfunctional requirement
- Epic
- Architectural epic
- Architectural requirement

While these may differ in terms of focus and scope, the basic concept is the same - the requirement, however named, expresses some intent or constraint the system must fulfill. This intent calls for work to be performed.

Sidebar: The troubled term "requirements"

In earlier times, the concept of "requirements" was often used as an sort of defense mechanism. Statements would often be heard such as:

"We can't start building anything, we don't fully understand the requirements"

"We can't change the requirements now, we've started building! Make up your mind!"

"The product is a failure because the business kept changing their mind about the requirements."

While the term "requirements" is still used throughout much education and training, the student should be aware of this history, and the fact that many Agile practitioners discourage use of the term.

As Jeff Patton says, "...I learned the word requirements actually means *shut up*." [\[Patton2014\]](#), introduction.

As systems engineering approaches transform to Cloud and infrastructure as code, a large and increasing percentage of IT work takes the form of altering symbolic files and moving them between states. We have seen this in the previous chapter, with artifacts such as scripts being created to drive the provisioning and configuring of computing resources.

Because the intent leads to the artifact, and the artifact leads to the commit, it makes sense to

associate the requirement with a branch in the version control system. This is not required, but makes it easier to trace the requirement to the actual work by which it was fulfilled. This will be discussed further in Section II and its associated labs.

User Story Mapping is a well known approach [Patton2014] with origins in the Scrum community. Here is an example from [Cohn2010]

"As a shopper, I can select how I want items shipped based on the actual costs of shipping to my address so that I can make the best decision."

The basic format is,

As a < type of user >, I want < goal >, so that < some value >.

The story concept is flexible, and can be aggregated and decomposed in various ways, as we will discuss in Chapter 4. Our interest here is in the basic stimulus for application development work that it represents.

You don't need an extensively automated system at this stage to capture requirements, but you need **something**. It could be a spreadsheet, or a shared word processing document, or sticky notes on a whiteboard (we'll talk about Kanban in the next section.) The important thing is to start somewhere, with team agreement as to what the approach is, so you can move forward collaboratively.

- INVEST
- MoSCoW

Test-driven development and refactoring

As a consequence of the introduction of new bugs, program maintenance requires far more system testing per statement written than any other programming. Theoretically, after each fix one must run the entire bank of test cases previously run against the system, to ensure that it has not been damaged in an obscure way. In practice such regression testing must indeed approximate this theoretical ideal, and it is very costly.

— Fred Brooks, *Mythical Man-Month*

The reason JUnit is important . . . is that the presence of this tiny tool has been essential to a fundamental shift for many programmers. A shift where testing has moved to a front and central part of programming. People have advocated it before, but JUnit made it happen more than anything else.

— Martin Fowler, <http://martinfowler.com/books/meszaros.html>

comment on Brooks

In test-driven development, the idea essence is to write code that tests itself. This is done through the creation of test harnesses and the tight association of tests with requirements. The logical culmination of test-driven development was expressed by Kent Beck in *extreme Programming*: write the test first [Beck2000]. Thus:

1. Given a “user story” (i.e [system intent](#)), figure out a test that will demonstrate its successful implementation.
2. Write this test using the established testing framework.
3. Write the code that fulfills the test.

Some readers may be thinking, “I know how to write a little code, but what is this about using code to write a test?”

While we avoid much in-depth examination of source code in this book, using some simplified Java will help. Here is an example drawn from the [Calavera project](#), the basis for the companion labs to this book.

Let’s say we want a function that will take a string of characters (e.g. a sentence) and wrap it in some [HTML](#) “Heading 1” tags. We will name the class “H1Class” and (by convention) we will start by developing a class called TestH1Class.

We write the test first:

```
public class TestClass1 {  
    private H1Class a; //  
    @Before  
    public void setUp() throws Exception {  
        this.a = new H1Class("TestWebMessage");  
    }  
    @Test  
    public void testTrue() {  
        assertEquals("string correctly generated",  
                    "<h1>TestWebMessage</h1>",  
                    this.a.webMessage());// string built correctly  
    }  
}
```

The code above basically states,

1. Set up the object to be tested
2. pass in a message with the content "TestWebMessage"
3. the test passes if we get back "<h1>TestWebMessage</h1>" - the original message surrounded by <h1> and </h1> "tags," which are part of HTML.

We run the test (e.g. through Junit and Ant, which we won’t detail here). It will fail.

Then, we write the class:

```
public class H1Class {  
    String strMsg;  
    public String webMessage()  
    {  
        return "<h1>" + strMsg + "</h1>";  
    }  
}
```

When we run the test harness correctly (e.g. using a build tool such as Ant or Maven), the test class will perform the following actions:

1. create an instance of the class H1Class, based on a string “TestWebMessage”
2. confirm that the returned string is “<h1>TestWebMessage</h1>”

If that string is not correctly generated, or the class cannot be created, or any other error occurs, the test fails and this is then reported via error results at the console, or (in the case of automated build) will be detected by the build manager and displayed as the build outcome.

Other languages use different approaches from that shown here, but every serious platform at this point supports test-driven development.

The associated course lab provides a simple but complete example of a test-driven development environment, based on lightweight virtualization.

Employing test-driven development completely and correctly requires thought and experience. But it has emerged as a practice in the largest scale systems in the world. Google runs many millions of automated tests daily [[Whittaker2012](#)]. It has even been successfully employed in hardware development [[Gruver2013](#)].

Refactoring

_Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations, each of which is "too small to be worth doing". However, the cumulative effect of each of these transformations is quite significant. By doing them in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time. _

— Martin Fowler, Refactoring -- <http://refactoring.com/>

Test-driven development enables the next major practice, that of refactoring. Refactoring is how

you address technical debt. What is technical debt? Technical debt is defined by Wikipedia as

...the eventual consequences of poor system design, software architecture or software development within a codebase. The debt can be thought of as work that needs to be done before a particular job can be considered complete or proper. If the debt is not repaid, then it will keep on accumulating interest, making it hard to implement changes later on. Unaddressed technical debt increases software entropy. Analogous to monetary debt, technical debt is not necessarily a bad thing, and sometime technical debt is required to move projects forward.

http://en.wikipedia.org/wiki/Technical_debt

Test driven development ensures that the system's functionality remains consistent, while refactoring provides a means to address technical debt as part of ongoing development activities.

Prioritizing the relative investment of repaying technical debt vs. developing new functionality will be examined in future sections, but at least you now know the tools and concepts.

Continuous integration

The term “continuous integration” was popularized by Paul Duvall in his book of the same name [\[Duvall2007\]](#).

In order to understand why continuous integration is important, it is necessary to further discuss the concept of source control and how it is employed in real world settings.

Imagine you have been working for some time with your partner in your startup (or on your small team) and you have three code modules. You are writing the web front end (file set A), your partner is writing the administrative tools and reporting (file set B), and you both partner on the data access layer (file set C). The conflict of course arises on the file set C that you both need to work on. A and B are mostly independent of each other, but changes to any part of C can have an impact on both your modules.

If changes are frequently needed to C, and yet you cannot split it into logically separate modules, you have a problem; you cannot both work on the same file at the same time. You also are concerned that the other person does not introduce changes into C that “break” the code in your module A.

In smaller environments, or under older practices, perhaps there is no conflict, or perhaps you can agree to take turns. But even if you are taking turns, you still need to test your code in A to make sure it's not been broken by changes your partner made in C.

And what if you really both need to work on C at the same time?

You might each choose to work on C on your own local copy. That way, you can move ahead on your local workstation. But when the time comes to combine both of your work, you will be in "merge hell." You may have chosen very different approaches to solving the same problem, and code may need massive revision to settle on one code base.

[TODO: add graphics]

These problems have driven the evolution of software configuration management for decades. In previous methods, to develop a new release, the code would be copied into a very long-lived branch. Ongoing “maintenance” fixes of the existing code base would also continue, and the two code bases would inevitably diverge. Switching over to the “new” code base might mean that once-fixed bugs (bugs that had been addressed by maintenance activities) would show up again, and of course this would not be acceptable.

So, when the newer development was complete, it would need to be merged back into the older line of code, and this was rarely if ever easy. In a worst case scenario, the new development might have to be redone.

Enter continuous integration. The key practices include:

- Source control (hopefully we have been harping on this enough that you are taking it seriously by now). Distributed version control systems such as git are especially popular, although older centralized products are [starting to adopt some of their functionality](#)
- Test-driven development
- Automated build activities
- Frequent integration of short-lived development branches with the main code repository (“mainline” or “trunk”)
- A defined package repository as a definitive location for the build output.

Current practices are well developed and represent a highly evolved understanding gained through the painful trial and error of many development teams over many years.

Rather than locking C so that only one person can work on it at a time, it's been found that the best approach is to allow developers to actually make multiple copies of such a file or file set and work on them simultaneously. Wait, you say. How can that work?

This is the principle of continuous integration at work. If the developers are continually pulling each other's work into their own working copies, and continually testing that nothing has broken, then distributed development can take place. So, if you are a developer, the day's work might be as follows:

8 AM: check out files from master source repository to a local branch on your workstation. Because files are not committed unless they pass all tests, you know that you are checking out clean code. You pull user story (requirement) that you will now develop.

8:30 AM: You define a test and start developing the code to fulfill it.

10 AM: You are closing in on wrapping up the first requirement. You check the source repository. Your partner has checked in some new code, so you pull it down to your local repository. You run all the automated tests and nothing breaks, so you're fine.

10:30: You complete your first update of the day; it passes all tests on your workstation. You commit it to the master repository. The master repository is continually monitored by the build server, which takes the code you created and deploys it, along with all necessary configurations, to a dedicated build server (which might be just a virtual machine or transient container). All tests pass

there (the test you defined as indicating success for the module, as well as a host of older tests that are routinely run whenever the code is updated).

11:00: Your partner pulls your changes into their working directory. Unfortunately, some changes you made conflict with some work they are doing. You briefly consult and figure out a mutually acceptable approach.

This is a continuously evolving problem area, with practices changing rapidly.

Continuous deployment

Finally, assuming that continuous integration is running effectively, one can take the last mile step and deploy the now tested and built software to pre-production or production environments.

At this point, the software can undergo user testing, load testing, integration testing, and so forth. Once those tests are passed, it can be deployed to production. (What is “production,” anyway? We’ll talk about environments in Section 2. For now, you just need to know that when an IT-based product is “in production,” that means it is live and available to its intended base of end users or customers.)

Moving new code into production has always been a risky procedure. Changing a running system always entails some uncertainty. However, the practice of infrastructure as code coupled with increased virtualization has reduced the risk. Often, a rolling release strategy is employed so that code is deployed to small sets of servers while other servers continue to service the load. This requires careful design to allow the new and old code to co-exist at least for a brief time.

Once software is compiled and built, the executable files that can be installed and run operationally should be checked into a [Package Manager](#).

This is important so that the versions of software used in production are well controlled and consistent. The package manager can then be associated with some kind of deploy tool that keeps track of what versions are associated with which infrastructure.

Timing varies by organization. Some strive for true “continuous deployment,” in which the new code flows seamlessly from developer commit through build, test, package and deploy. Others put gates in between the developer and check-in to mainline, or source to build, or package to deploy, so that some human governance remains in the toolchain. We will go into more detail on these topics in chapters 6, 9, and 10.

Future trends in AD

- Microservices

[this section needs to be written]

add discussion of patterns/antipatterns and link throughout.

Conclusion

Applications are why computers exist. Supporting applications increasingly is less about systems engineering, and more about quickly provisioning standard, shared infrastructure. Application development has moved decisively in the past twenty years to Agile delivery models, based on techniques such as:

- story mapping
- test-driven development
- refactoring
- continuous integration
- continuous deployment

Application delivery, software development, and the Agile movement are broad, complex, and evolving topics. For those of you familiar with Agile, we have only scratched the surface in this chapter.

In future chapters we will go into more detail on topics such as:

- Product management, including behavior-driven development and continuous design (much more on requirements, user stories, etc)
- The importance of feedback
- Prioritization and cost of delay
- Scrum and Kanban
- Tracking tasks and effort
- Closing the loop from operations to development, and coping with interrupt-driven work

and much more.

Discussion questions

- What is your exposure to application programming?
- Can you think of examples of waterfall and agile approaches in your daily life (not necessarily related to IT?)

Research & practice

- Review Amazon's [AWS CodePipeline](#)
- Download the [Calavera project](#) and run it on your laptop or workstation. [Instructions](#).

Further reading

Books

- Martin Fowler, [Refactoring](#)
- McConnell, Steve, [Code Complete](#).

- Jez Humble & David Farley, *Continuous Delivery*
- Gene Kim et al, *Phoenix Project*
- Paul Duvall et al, *Continuous Integration*
- Michael Nygard, *Release IT*

Articles

- [Introducing the Software Testing Cupcake \(Anti-Pattern\)](#)
- [Release Testing is Risk Management Theater](#)
- [Continuous Integration is Dead](#)
- [Abandon your DVCS and Return to Sanity](#)

Section I Conclusion

We are now at the end of the first section, and one quarter the way through this book (and this course, assuming you are taking a semester long treatment).

You are now the proud master or mistress of a functioning startup. You have decided to provide some product that at least partially depends on an IT-based component that you need to actively develop.

You understand the value that IT brings, and your own product's needs for it. You have chosen at least a functioning platform for initial development, without falling into the trap of analysis paralysis, although at this point you should be keeping your options open if your initial platform choice doesn't prove out.

Finally, you have implemented at least a lightweight continuous delivery pipeline. You didn't need to spend any money doing this, as so much powerful technology is freely available. In particular, from the start you have taken version control very seriously and have a stable, backed up source repository as a basis for your product development.

You also have at least rudimentary systems for tracking requirements, building your software, storing your packages, and deploying them to your production environment.

Congratulations, you've got all the basics in place. Your product is starting to attract sales and/or investors, and you've hired a few more people. Let's talk about collaboration.

Staying at Level 1 In your career, many - perhaps even the majority - of the people you meet and work with will be focused on Level 1 in their thinking and approach. This is a fine thing and to be expected. Level 1 is where the real work is done.

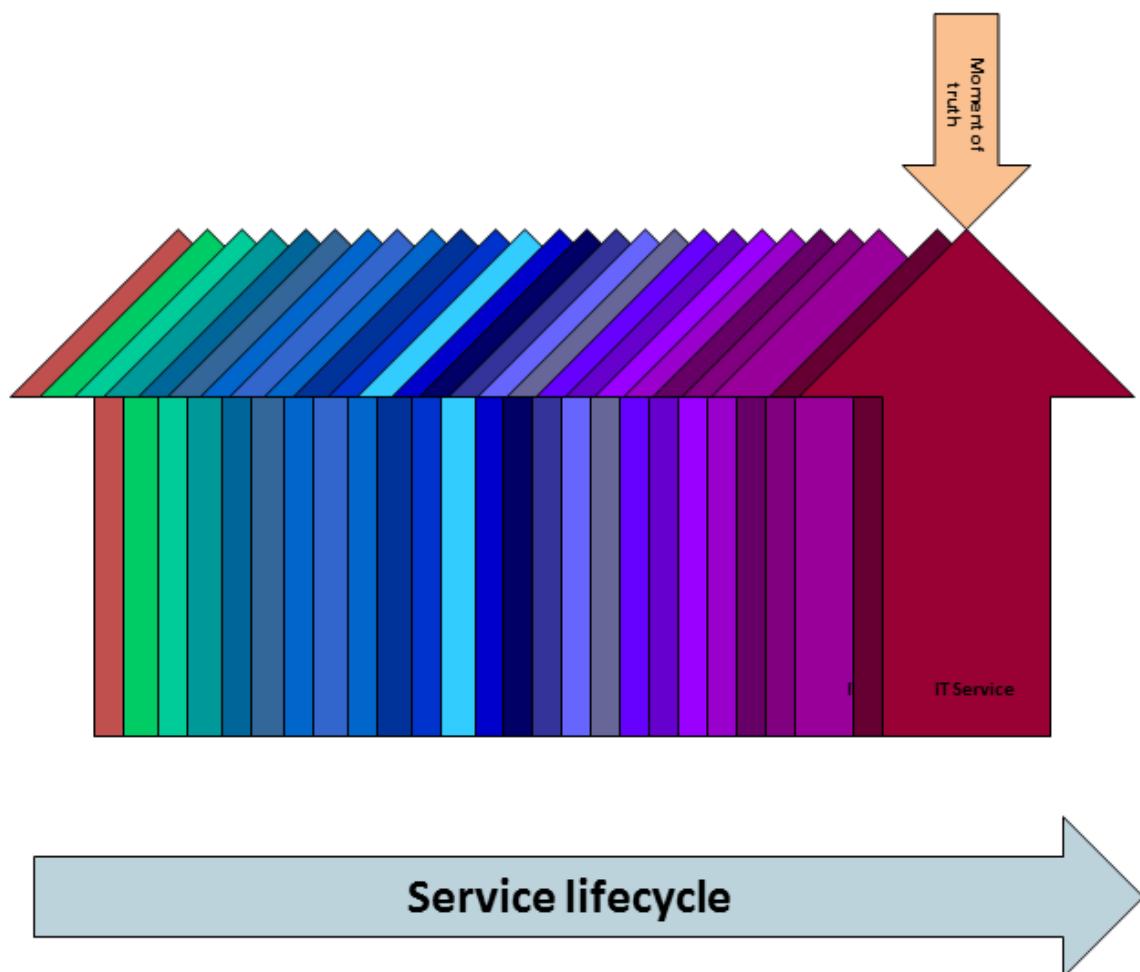
Section II: Team

Section II introduction

Scenario You are part of a tight-knit team. You've moved out of the garage into a more professionalized environment. You might not all be in the same room, but you have a single mission and a cohesive culture, and you don't need a lot of overhead to get the job done.

Your startup has met with some success and you are now a team - still small enough to be fed by two pizzas—if people aren't that hungry. Even with a few new people comes the need to more clearly establish your product direction, so people are building the right thing. You're all in the same location, and can still communicate informally, but there is enough going on that you need a more organized approach to getting work done. Finally, this great thing you're building doesn't mean much if people cannot understand how best to use it, or if it's not running and right people can't get to it.

However, things are getting larger and more complex. You have a significant user base, and your founder is increasingly out meeting with users, customers and investors; she isn't in the room with the product team as much any more... in fact, she just named someone to be "product owner" and what is that all about?



Remember in Chapter 1 we talked of the user's **value experience**, and also how services **evolve over**

time in a lifecycle?

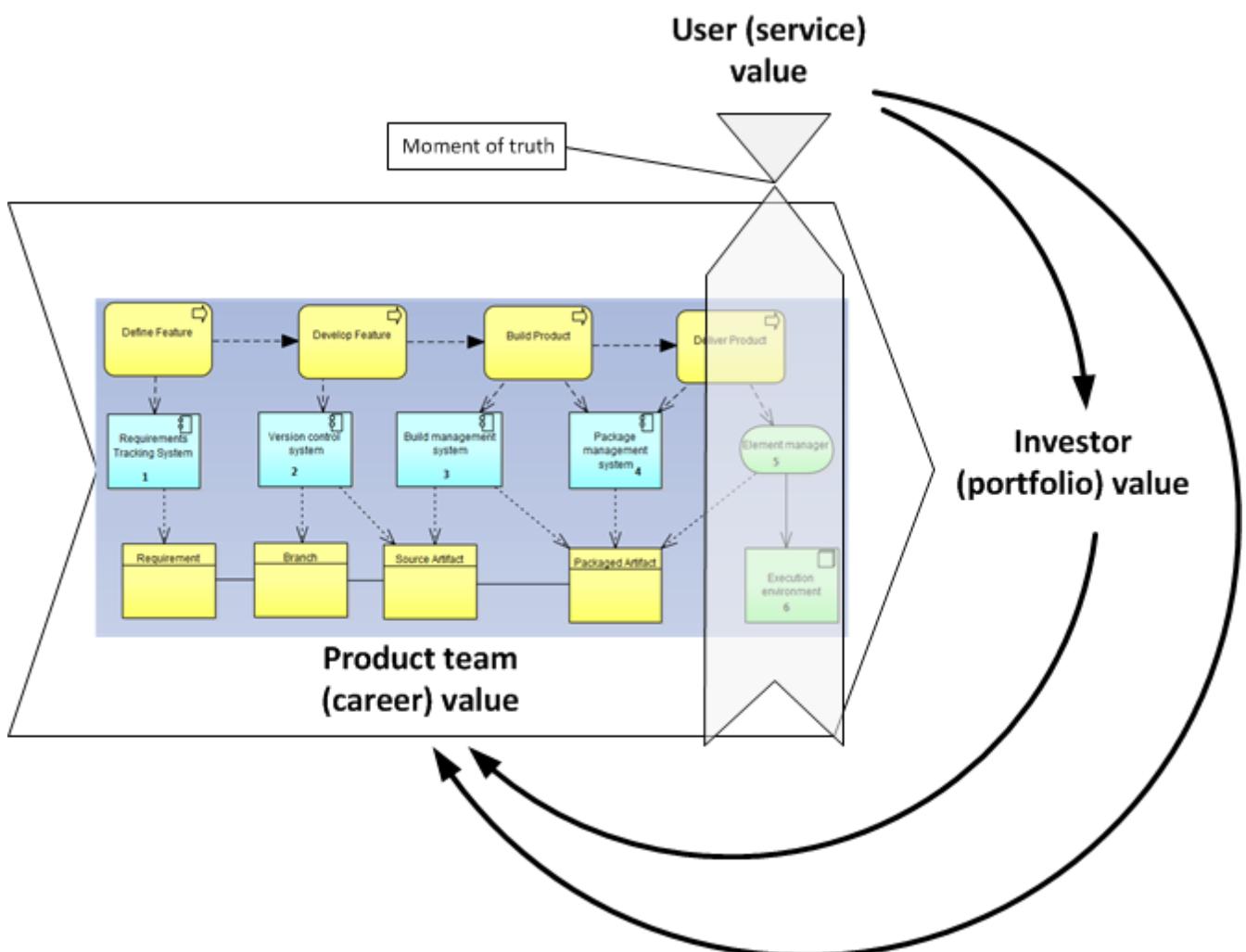
There are two primary value experiences:

- The value the user derives from the service (e.g. account lookups, or a flawless navigational experience)
- The value the investor derives from monetizing the product, or comparable incentives (e.g. nonprofit missions)

Additionally, the product team derives career value. This becomes more of a factor later in the game. We will discuss this further in chapter 7 on organization, and Chapter 11 on architecture lifecycles & technical debt.

The product team receives feedback from both value experiences. The day to day interactions with the service (e.g. help desk and operations) are understood, and (typically on a more intermittent basis) the portfolio investor also feeds back the information to the product team (the boss's boss comes for a visit).

This is of course a simplified model, and in a large enterprise both the users and investors may have proxies (more on this to come).



The figure above is an architectural representation of the toolchain we started to develop in Section 1. The toolchain can be seen as part of a value chain that is delivering an IT service.

In this section, we will look at:

- How you are accepting and executing on feedback signals
- How the feedback relationship with your investors is evolving, in terms of your product direction
- How the feedback relationship with your users is evolving, in terms of both operational criteria and product direction

We will discuss work management, product management, shared mental models, visualization, and systems monitoring.

The practices and approaches established at the team level are critical to the higher levels of scale discussed in Sections 3 and 4. In this section, we discuss small, cross-functional, outcome-oriented teams. We discuss collaboration and customer intimacy, and the need to limit work in process. And we discuss blameless cultures where people are safe to fail and learn.

If these learnings are not firmly established, the organization may struggle to scale; the need for cross-team coordination can (ironically) destroy the performance of individual teams, if not handled well. Scaling success starts with building a strong team level.

Chapter 4: Product Management

You (as the startup leader) are spending more time with investors and customers, and maintaining alignment around your original product vision is becoming more challenging as you are pulled in various directions. You need some means of keeping the momentum here. And the concept of "product management," you're finding, represents a rich set of ideas for managing your team's efforts at this stage of the game.

Chapter 5: Work Management

Even with a small team of 5 people (let alone 8 or 9), it's too easy for balls to get dropped as work moves between key contributors. You probably don't need a complex software-based process management tool yet, but you do need some way of managing work in process. And you start to understand that work takes many forms and exists as a concept at different scales.

Chapter 6: Operations Management

Since Chapter 3, your application developers have been running your systems and even answering the occasional phone call from customers. You're big enough that you need a bit more specialization. You've got dedicated support staff answering the phone calls and you are finding that, even if you rotate operational responsibilities across developers, it is still a distinct kind of "interrupt-driven" work that is not compatible with heads-down, focused software development. You've probably seen by now that complex systems are fragile and tend to fail; how you learn (or don't) from those failures is a critical question.

Special section: Systems thinking and feedback

"The harder you push, the harder the system pushes back."

— Peter Senge, The Fifth Discipline

A brief introduction to feedback

While the term "information systems (IS)" was widely replaced by "information technology (IT)" in the 1990s, do not be fooled. Enterprise IT is a complex [sociotechnical system](#), that delivers the digital services to support a myriad of other complex sociotechnical systems.

Merriam-Webster [defines a system](#) as "a regularly interacting or interdependent group of items forming a unified whole". These interactions and relationships quickly take center stage as you move from individual work to team efforts. Consider that while a 2 member team only has 1 relationship to worry about, a 10 member team has 45, and a 100 person team has 4,950!

As the Senge quote implies, brute force does not scale well within the context of a system. Systems tend to make their own rules. Within the bounds of the system, actions lead to outcomes, which in turn affect future actions. This **feedback** often becomes amplified and re-amplified through the systems' interactions with itself, until an overload occurs, such as the horrible screeching of a microphone too close to a speaker. In the business world, these feedback loops can manifest themselves in rigid platforms, brittle applications, or even a toxic work culture that sabotages success.

A thorough discussion of systems theory is beyond the scope of this book. However, many of the ideas that follow are informed by it. Obtaining a working knowledge of systems theory will not only enhance your understanding of this book, it can also be an essential tool for managing uncertainty in your future career, teams, and organizations.

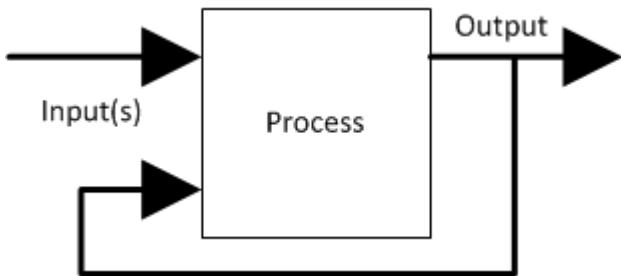
reference Meadows & Weinberg

Defining feedback, with examples

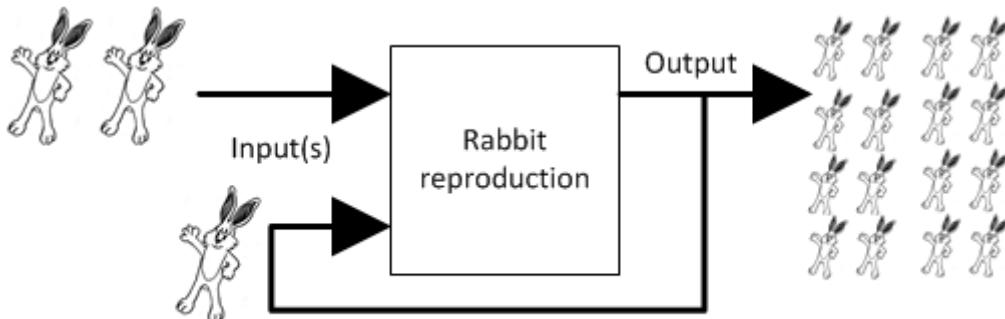
Feedback is a loaded term. We hear terms like positive feedback and negative feedback, and associate it quickly with performance coaching and management discipline. That is not the sense of feedback in this book.

The definition of feedback as used in this book is based on engineering. There is considerable related theory in general engineering and especially [control theory](#) and the reader is encouraged to investigate some of these foundations if unfamiliar.

Here is the classic illustration of a feedback loop:



For example, we can consider “rabbit reproduction” as a process with a positive feedback loop:

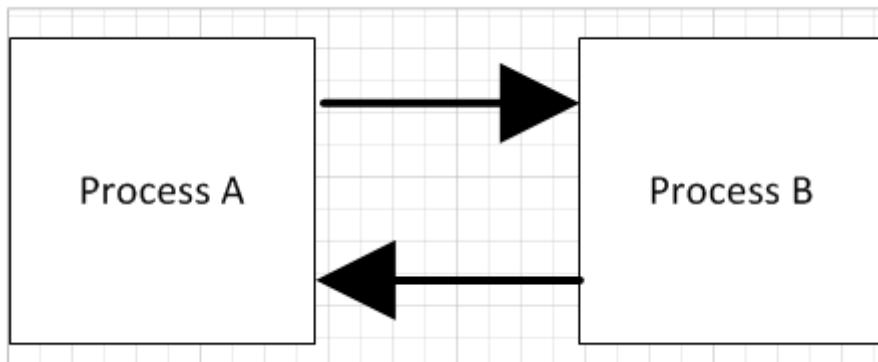


A “positive” feedback loop

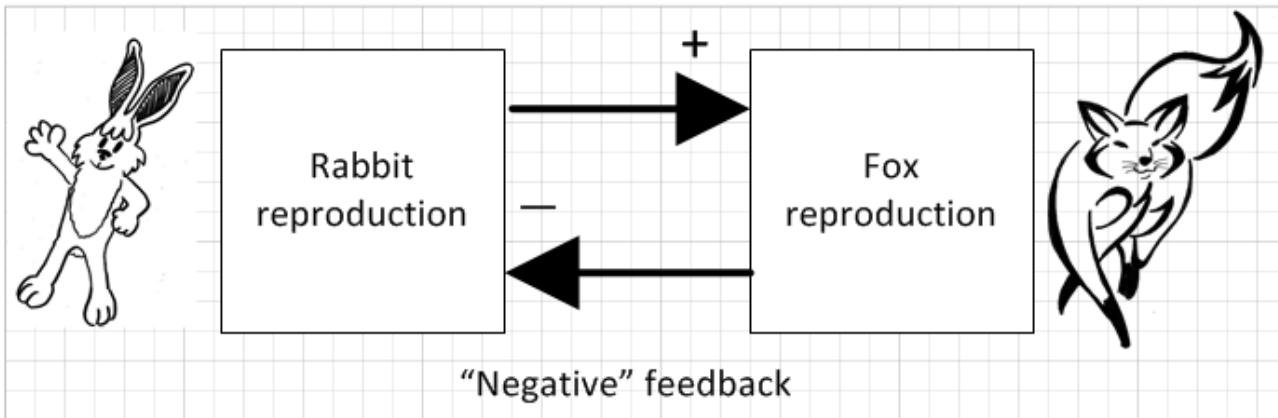
The more rabbits, the faster they reproduce, and the more rabbits.

This is a “positive” feedback loop, although Mr. MacGregor the local gardener may not agree, given that they are eating all his cabbages!! This is why feedback experts (e.g. [\[Sterman2000\]](#)) prefer to call this “reinforcing” feedback, because there is not necessarily anything “positive” about it.

We can also consider feedback as the relationship between TWO processes:



In our rabbit example, what if Process B is fox reproduction, that is, the birth rate of foxes (who eat rabbits)?



More rabbits equals more foxes (notice the “+” symbol on the line), because there are more rabbits to eat! But what does this do to the rabbits? It means LESS rabbits. Which, ultimately, means less foxes... and at some point, the populations balance. This is classic negative feedback. However, the local foxes don't see it as negative (nor do the local gardeners!) That is why feedback experts prefer to call this “balancing” feedback.

[CLD links ANI] | https://upload.wikimedia.org/wikipedia/commons/d/d8/CLD_links_ANI.gif

Linked from [Wikipedia](#)

Wikipedia has good articles on [Causal Loop Diagramming](#) and [Systems Dynamics](#) (with cool dynamic visuals.) [\[Sterman2000\]](#) is the definitive text with applications.

What does systems thinking have to do with IT?

In an engineering sense, positive feedback is often dangerous and a topic of concern.

The classic example of bad positive feedback in engineering is the collapse of the Tacoma Narrows bridge, "Galloping Gertie."



[Video](#)

Linked from [Wikipedia](#).

As with bridges, at a technical level, reinforcing feedback can be a very bad thing in IT systems. In general, any process that is amplified without any balancing feedback will eventually consume itself and all its resources, just like rabbits will eat all the food available to them. So, if you create a process (e.g. write and run a computer program) that recursively spawns itself, it will sooner or later crash the computer as it devours memory and CPU. See [Runaway processes](#).

Balancing feedback, on the other hand, is critical to make sure you are “staying on track.” Engineers use concepts of [control theory](#), for example [damping](#), to keep bridges from falling down. Balancing feedback in a business and IT context takes a wide variety of forms:

- The product owner clarifying for developers the precise user experience vision for the product, based on a demonstration of developer work in process
- The end users calling to tell you the “system is slow” (or down)
- The product owner or portfolio sponsor calling to tell you they are not satisfied with the system’s value

In short, we see these two basic kinds of feedback:

- Positive/Reinforcing, “do more of that”
- Negative/Balancing, “stop doing that,” “fix that”

NOTE

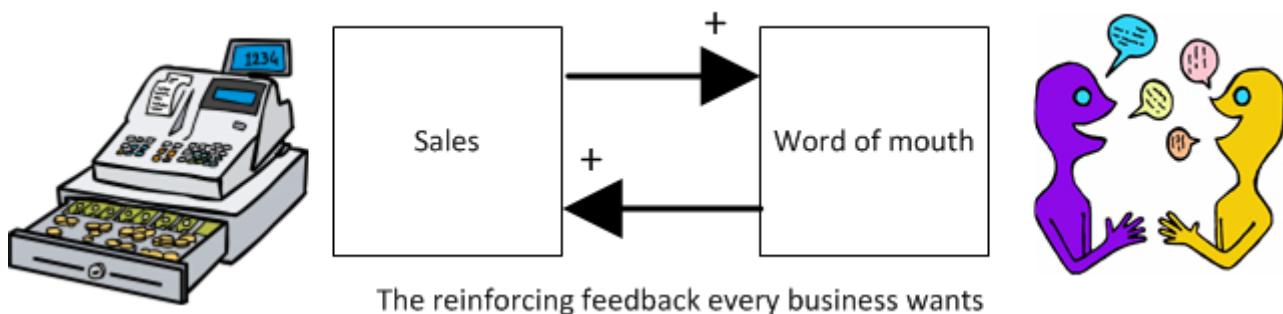
Still confused? Think about the last time you saw a "reply-all" email storm. The first accidental mass send generates feedback (emails saying "take me off this list"), which generate more emails ("stop emailing the list") and so on. This does not continue indefinitely; management intervention, common sense and fatigue eventually damp the storm down.

Positive feedback: the special case investors want

[Adoption SFD ANI s] |

Linked from [Wikipedia](#)

At a business level, there is a special kind of positive feedback that defines the successful business:



This is reinforcing feedback and positive for most people involved: investors, customers, employees.

At some point, if the cycle continues, it will run into balancing feedback:

- Competition
- Market saturation
- Negative externalities (regulation, pollution, etc)

But those are the problems the business wants to have.

Open vs closed loop systems

Finally, we should talk briefly about open loop versus closed loop systems.

- Open loop systems have no regulation, no balancing feedback
- Closed loop systems have some form of balancing feedback

In navigation terminology, the open-loop attempt to stick to a course without external information (e.g. navigating in the fog, without radar or communications) is known as "[dead reckoning](#)," in part because it can easily get you dead!

A good example of an open loop system is the children's game "pin the tail on the donkey." In "pin the tail on the donkey," a person has to execute a process (pinning a paper or cloth "tail" onto a poster of a donkey - no live donkeys are involved!) while blindfolded, based on their memory of their location (and perhaps after being deliberately disoriented by spinning in circles).

Since they are blindfolded, they have to move across the room and pin the tail without the ongoing corrective feedback of their eyes. (Perhaps they are getting feedback from their friends, but perhaps their friends are not reliable....)



Photo Credit - [mike krzeszak, Flickr, Creative Commons](#)

Without the blindfold, it would be a closed loop system. The person would rise from their chair and, through the ongoing feedback of their eyes to their central nervous system, would move towards the donkey and pin the tail in the correct location.

This may seem obvious, but the history of IT management (some would say all management) over the past decades has been the struggle to overcome open-loop practices. Reliance on open-loop practices is arguably an indication of a dysfunctional command and control culture. A IT team that is designing and delivering without sufficient corrective feedback from its stakeholders is an ineffective, open-loop system. [\[Kennaley2010\]](#) applies these principles to software development in much greater depth, and is recommended.

NOTE

No system can ever be fully "open loop" indefinitely. Sooner or later, you take off the blindfold, or wind up on the rocks. Mark Kennaley prefers the term "delayed feedback closed loop system" (personal correspondence, 9/2015).

Engineers of complex systems use feedback techniques extensively. Complex systems do not work without them. **This section is about closing the loops.**

Quality management

Deming and Juran.

Chapter 4: Product Management

Collaborative

Status: This chapter is in "draft 1.5" - improved from original first draft completed in 2015.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Introduction to Chapter 4

Product Management?

"Product management?" In a book on IT management? Those of you with industry experience, especially backgrounds in project-based enterprise software development, may be unfamiliar with the term. However, a focus on product development is one of the distinguishing features of Agile development, even if that development is taking place in a larger enterprise context.

As you grow your company, you are bringing more people in. You become concerned that they need to share the same vision that inspired you to create this company. This is the goal of product management as a formalized practice.

Product strategy was largely tacit in Section I. As the founder, you used product management and discovery practices, and may well be familiar with the ideas here, but the assumption is that you did not explicitly **formalize** your approach to them. Now you need a more prescriptive and consistent approach to discovering, defining, designing, communicating, and executing a product vision across a diverse team.

In this chapter, we will define and discuss product management, and distinguish it from project and process management. We will cover how product teams are formed and what practices and attitudes you should establish quickly.

We will discuss a number of specific schools of thought and practices, including Gothelf's Lean UX, Scrum, and more specific techniques for product "discovery." Finally, we will discuss the concepts of design and design thinking.

Chapter 4 outline

- Why product management?
 - The product vision
 - Defining Product Management
 - Process, project, and product management
 - Productization as a strategy at Amazon
- Organizing the product team
 - The concept of collaboration
 - Lean UX
 - Scrum
 - More on product team roles
- Product discovery
 - Formalizing product discovery
 - Product discovery techniques
 - Discovery and design
 - Design
- Assorted topics in Product Management

Chapter 4 learning objectives

- Define and distinguish product versus project and process management
- Identify the key concerns of forming a collaborative product team
- Describe current product-oriented practices, such as Lean UX and Scrum
- Describe product design and discovery practices and concerns

Why product management?



Product design session [8: Image credit <https://www.flickr.com/photos/daonb/6223628837>, downloaded 2016-09-14, commercial use permitted]

The product vision

NOTE You should review the [digital context](#) material in Chapter 1

Before work, before operations, there must be a vision of product. You already established a preliminary vision in Chapter 1, but now as your organization grows, you need to consider further how you will sustain that vision and establish an ongoing capability for realizing it.

Like many other topics in this book, product management is a significant field in and of itself. Historically, product management has **not** been a major theme in enterprise IT management. Digital changes this.

IT systems started by serving narrow purposes, often “back office” functions such as accounting or [materials planning](#). Mostly, such systems were managed as projects assembled on a temporary basis, resulting in the creation of a system to be “thrown over the wall” to operations.

Product management, on the other hand, is concerned with the entire lifecycle. The product manager cares about the vision, its execution, the market reaction to the vision (even if an internal market), the health, care and feeding of the product, and the product’s eventual sunset or replacement.

In the enterprise IT world, "third party" vendors (e.g. IBM) providing the back office systems had product management approaches, but these were **external** to the IT operations.

Nor were IT-based product companies as numerous forty years ago as they are today; as noted in chapter 1, the digital component of modern products [continues to increase](#) to the point where it's often not clear whether a product is "IT" or not.

Reacting to market feedback and adapting product direction is an essential role of the product owner. In the older model, feedback was often unwelcome, as the project manager typically was committed to the [open-loop dead reckoning](#) of the project plan and changing scope or direction was seen as a failure, more often than not.

Now, it's accepted that systems evolve, perhaps in unexpected directions. Rapidly testing, failing fast, learning, and pivoting direction are all part of the lexicon, at least for market-facing IT-based products. And even back-office IT systems with better understood scope are being managed more as systems (or products) with lifecycles, as opposed to transient projects. (See the [Amazon discussion](#), below.)

So, what is product management and what does it mean for your team?

Defining Product Management

In order to define product management, we first need to define product. In Chapter 1, we established that products are goods, services, or some combination, with some feature that provides value for some consumer. BusinessDictionary.com [defines it thus](#):

[A Product is] A good, idea, method, information, object or service created as a result of a process and serves a need or satisfies a want. It has a combination of tangible and intangible attributes (benefits, features, functions, uses) that a seller offers a buyer for purchase. For example a seller of a toothbrush not only offers the physical product but also the idea that the consumer will be improving the health of their teeth . . . A good or service [must] closely meet the requirements of a particular market and yield enough profit to justify its continued existence.

— BusinessDictionary.com

Product *management*, according to the [same source](#), is

The organizational structure within a business that manages the development, marketing and sale of a product or set of products throughout the product life cycle. It encompasses the broad set of activities required to get the product to market and to support it thereafter.

— BusinessDictionary.com

Product management in the general sense often reports to the Chief Marketing Officer (CMO). It represents the fundamental strategy of the firm, in terms of its value proposition and viability. The product needs to reflect the enterprise's strategy for creating and maintaining customers.

Product strategy for **internally-facing** products is usually **not** defined by the enterprise CMO. If it is a back-office product, then "business within a business" thinking may be appropriate. (Even the payroll system run by IT for HR is a "product," in this view.) In such cases, there still is a need for someone to function as an "internal CMO" to the external "customers."

NOTE

As a field, product management has a professional association, the [Product Development and Marketing Association](#), which publishes an extensive and continuously-refined handbook, and supports local chapters, training and certification, and other activities typical of a mature professional organization.

With [digital transformation](#), all kinds of industrial products have increasing amounts of "IT" in them. This means that an understanding of IT, and ready access to any needed IT specialty skills, is increasingly important to the general field of product management.

Product management includes research and development, which means that there is considerable uncertainty. This is of course also true of IT systems development.

Perhaps the most important aspect of product design is focusing on the user, and what she or he needs. The concept of **outcome** is key. This is easier said than done. The general problem area is considered Marketing, a core business school topic. Entire books have been written on the various tools and techniques for doing this, from focus groups to ethnographic analysis.

However, Marty Cagan warns against confusing Product Management and Product Marketing. He defines the two as follows:

The product manager is responsible for defining—in detail—the product to be built, and validating that product with real customers and users. The product marketing person is responsible for telling the world about that product, including positioning, messaging and pricing, managing the product launch, providing tools for the sales channel to market and sell the product, and for leading key programs such as online marketing and influencer marketing programs. [Cagan2008], pp. 10-11.

We discuss some criticisms of overly marketing-driven approaches [below](#).

Process, project, and product management

In the remainder of this book, we will continually encounter three major topics:

- Product Management (Chapter 4)
- Project Management (covered in Chapters 8 and 9)
- Process Management (covered in Chapter 9)

They have an important commonality: **all of them are concepts for driving results across**

functional or skill silos.

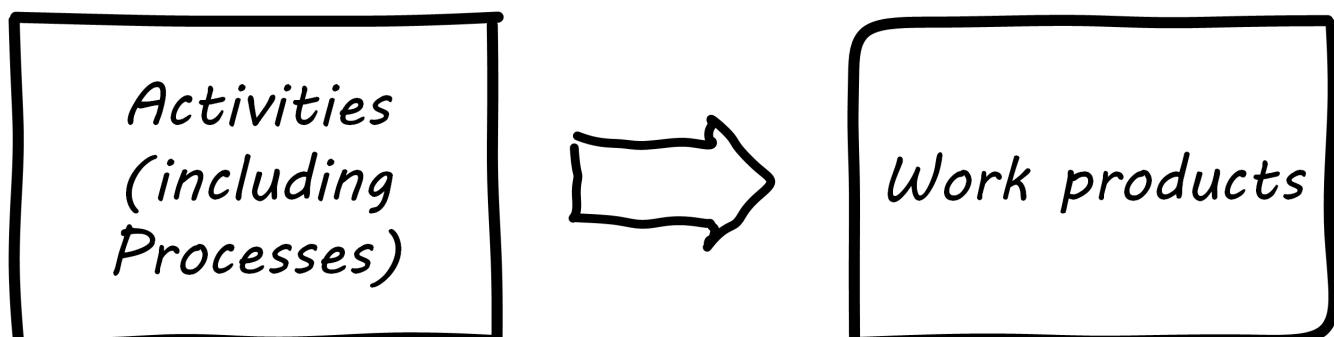
Here are some of the key differences between process, project and product management in the context of digital services and systems:

Process	Project	Product
Task oriented	Deliverable oriented	Outcome oriented
Repeatable with a high degree of certainty	Executable with a medium degree of certainty	Significant component of research and development, less certain of outcome - empirical approaches required
Fixed time duration, relatively brief (weeks/months)	Limited time duration, often scoped to a year or less	No specific time duration; lasts as long as there is a need
Fixed in form, no changes usually tolerated	Difficult to change scope or direction, unless specifically set up to accommodate	Must accommodate market feedback and directional change
Used to deliver service value and operate system (the "Ops" in DevOps)	Often concerned with system design and construction, but typically not with operation (the "Dev" in DevOps)	Includes service concept and system design, construction, operations, and retirement (both "Dev" and "Ops")
Process owners concerned with adherence and continuous improvement of process. Otherwise can be narrow in perspective.	Project managers are trained in resource and timeline management, dependencies & scheduling. They are not typically incented to adopt a long-term perspective.	Product managers need to have project management skills as well as understanding market dynamics, feedback, building long-term organizational capability
Resource availability and fungibility is assumed	Resources are specifically planned for but their commitment is temporary (team is "brought to the work")	Resources are assigned long-term to the product (work is "brought to the team")

The above distinctions are deliberately exaggerated and there are of course exceptions (short projects, processes that take years). However, it is in the friction between these perspectives we see some of the major problems in modern IT management.

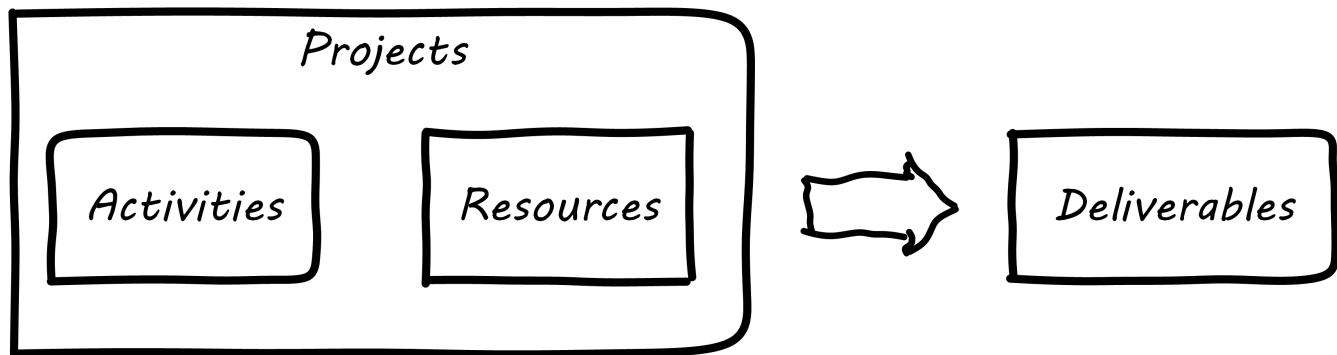
Here is another way to explain it:

An activity, which may be a one-time task or a repeatable process, results in some Work Product:



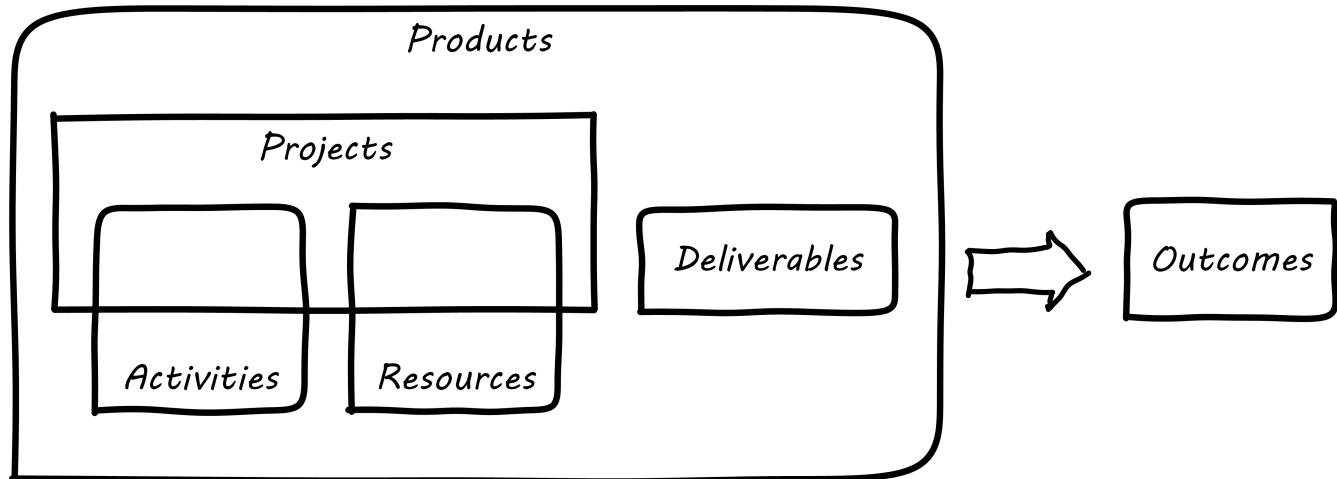
The consumer or stakeholder of that work product might be a Project Manager.

Project management includes concern for both the activities, AND the resources (people, assets, software) required to produce some deliverable:

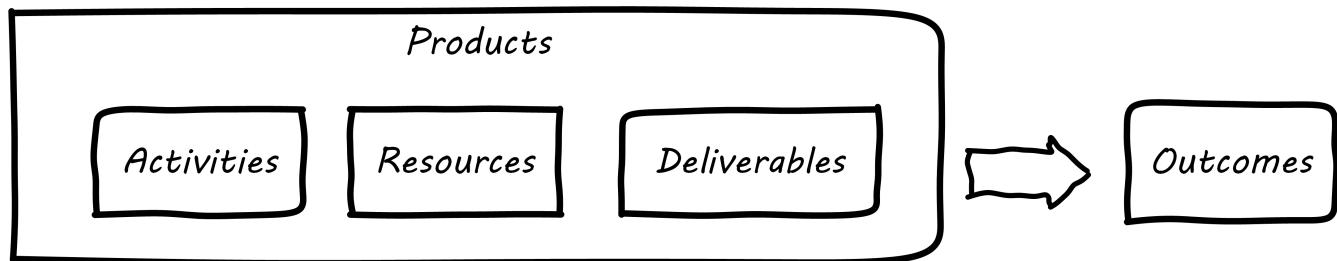


The consumer of that deliverable might be a Product Manager.

Product management includes concern for projects and their deliverables, and their ultimate **outcomes** (either in the external market or internally):



Notice that product management may directly access activities and resources. In fact, earlier-stage companies often do not formalize project management.



In our scenario, you are now on a tight-knit, collaborative team. It is preferable that you think in terms of developing and sustaining a product. However, projects still exist, and sometimes you may find yourself on a team that is funded and operated on that basis. We will return to projects and formal process management in Chapters 8 and 9.

Productization as a strategy at Amazon

Amazon (the online bookseller) is an important influence in the modern trend towards product-centric IT management. First, the founder Jeff Bezos mandated that all software development should be [service-oriented](#), and assume that the functionality being built might at some point be offered to external customers.



Can you feed your team with two pizzas? [9: Image credit <https://www.flickr.com/photos/ramblinbears/7937873272>, downloaded 2016-09-20, commercial use permitted]

Second, a widely reported practice at Amazon.com is the [limitation of product teams to between 5-7 people](#), the number that can be fed by “two pizzas” (depending on how hungry they are).

There are [good reasons](#) for this approach. It has long been recognized in software and IT management that larger teams do not necessarily result in higher productivity. The best known statement of this is Fred Brooks’ rule from *The Mythical Man-Month*, that “adding people to a late project will make it later.”[\[Brooks1975\]](#)

NOTE

Fred Brooks' *The Mythical Man-Month*, derived in part from his experiences leading the IBM OS-360 project, is one of the timeless classics in software engineering and IT management writing. Serious IT professionals, whether or not they are actually programmers, should have it on their bookshelves.

The reasons for this phenomenon have been studied and analyzed (see e.g. [\[Madachy2008\]](#)) but in general it is due to the increased communication overhead of expanded teams.

Product design work (of which software development is one form) is creative and highly dependent on tacit knowledge, interpersonal interactions, organizational culture, and other “soft” factors. Products, especially those with a significant IT component, can be understood as socio-technical systems, often complex. This means that small changes to their components or interactions can have major effects on their overall behavior and value.

This in turn means that newcomers to a product development organization can have a profound impact on the product. Getting them “up to speed” with the culture, mental models, and tacit assumptions of the existing team can be challenging and rarely is simple. And the bigger the team, the bigger the problem.

The net result of these two practices at Amazon (and now [General Electric](#) and many other companies) is the creation of multiple nimble services that are decoupled from each other, constructed and supported by teams appropriately sized for optimal high-value interactions.

Organizing the product team

You are a team now. Your founder and co-founder have found enough interest to sustain a larger organization.

How are you going to organize? How are you going to work? Events move quickly, and you don’t have much time to think about these things. But getting things right at the team level is essential as your organization scales up. Bad habits (like accepting too much work in the system, or tolerating toxic individuals) will be more and more difficult to overcome as you grow.

The concept of collaboration

Individuals and interactions over processes and tools.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

— Agile Manifesto

IMPORTANT

We will discuss this in more depth in Chapters 7 and 8. But this chapter is the first discussion of "how are we with each other." Culture requires attention at the earliest stages, as it can be very difficult to change later.

Team collaboration is one of the key values of Agile. The Agile Alliance states that

A "team" in the Agile sense is a small group of people, assigned to the same project or effort, nearly all of them on a full-time basis.

Teams are multi-skilled, share accountability, and individuals on the team may play multiple roles. [\[Alliance2015\]](#)

Face to face interactions, usually enabled by giving the team its own space, are seen as essential for collaboration. However, collaboration does not happen just because people are fed pizzas and work in a room together. Google has established that the most significant predictor of team performance is a sense of psychological safety (see sidebar).



Psychological safety is essential to collaboration [10: Image credit <https://www.flickr.com/photos/marckjerland/4633544440>, downloaded 2016-09-20, commercial use permitted]

Sidebar: Google's Project Aristotle

Around 2012, Google became interested in answering the question:

What makes a Google team effective?

Based on 200+ interviews across 180+ teams, they determined that "Who is on a team matters less than how the team members interact, structure their work, and view their contributions."

They identified five "key dynamics":

1. Psychological safety: team members feel safe to take risks with each other
2. Dependability: team members can be counted on
3. Structure and clarity: roles, plans and goals are clear
4. Meaning of work: work is personally important
5. Impact of work: the work matters

Of the 5, psychological safety was the most significant. Teams that cultivate this enable collaboration and creativity, which lead to product value and improved organizational performance. [\[Rozovsky2015\]](#)

We turn to two current schools of thought with much to say about collaboration: Lean UX and Scrum.

Lean UX

Lean UX is the practice of bringing the true nature of a product to light faster, in a collaborative, cross-functional way that reduces the emphasis on thorough documentation while increasing the focus on building a shared understanding of the actual product experience being designed.

— Jeff Gothelf, Lean UX

Lean UX is a term coined by author and consultant Jeff Gothelf [\[Gothelf2013\]](#), which draws on three major influences:

- Design thinking
- Agile software development
- Lean Startup

We briefly discussed [Lean Startup](#) in Chapter 1, and the history and motivations for [Agile software development](#) in Chapter 3. We'll look in more depth at product discovery techniques, and [design and design thinking](#) in the next chapter section. However, Lean UX has much to say about forming the product team, suggesting (among others) the following principles for forming and sustaining teams:

- Dedicated, cross-functional teams
- Outcome (not deliverable/output) focus
- Cultivating a sense of shared understanding
- Avoiding toxic individuals (so-called "rockstars, gurus, and ninjas")
- Permission to fail

(Other Lean UX principles such as small batch sizes and visualizing work will be discussed elsewhere; there is significant overlap between Lean UX and other schools of thought covered in this book.)

Lean UX is an influential work among digital firms and summarizes modern development practices well, especially for small, team-based organizations with minimal external dependencies. It is a broad and conceptual, principles-based framework open for interpretation in multiple ways. We continue with more "prescriptive" methods and techniques, such as Scrum.

Scrum

Scrum is a lightweight framework designed to help small, close-knit teams of people develop complex products.

— Chris Sims/Hillary L. Johnson, Scrum: A Breathtakingly Brief and Agile Introduction

There Are No Tasks; There Are Only Stories.

— Jeff Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time

One of the first prescriptive Agile methodologies you are likely to encounter as a practitioner is Scrum. There are many books, classes, and websites where you can learn more about this framework; [\[Sims2012\]](#) is a good brief introduction, and [\[Rubin2012\]](#) is well suited for more in-depth study.

"Prescriptive" means detailed and precise. A doctor's prescription is specific as to what medicine to take, how much, and when. A prescriptive method is similarly specific. "Agile software development" is not prescriptive; as currently published by the Agile Alliance, it is a collection of principles and ideas you may or may not use.

NOTE

By comparison, Scrum is prescriptive; it states roles and activities specifically and trainers and practitioners in general seek to follow the method completely and accurately.

Scrum is appropriate to this chapter, as it is product-focused. It calls for the roles of:

- Product owner
- Scrum master
- Team member

and avoids further elaboration of roles.

The Scrum product owner is responsible for holding the product vision and seeing that the team executes the highest value work. To do this, the potential features for the product are maintained in a "backlog" that can be re-prioritized as necessary (rather than a large, fixed-scope project). The product owner also defines acceptance criteria for the backlog items.

The Scrum Master, on the other hand, acts as a team coach, "guiding the team to ever-higher levels of cohesiveness, self-organization, and performance." [\[Sims2012\]](#).

Scrum uses specific practices and artifacts such as sprints, standups, reviews, the above-mentioned concept of backlog, burndown charts, and so forth. We will discuss these in Chapter 5 (Work Management) along with Kanban, another popular approach for executing work.

- There are three roles:
 - The product owner sets overall direction.
 - The Scrum Master coaches and advocates for the team
 - The development team is defined as those who are committed to the development work
- There are various artifacts:
 - Product backlog is the overall "to-do" list for the product. (More on product management in the next chapter.)
 - Sprint backlog is the to-do list for the current sprint
 - Potentially shippable product increment is an important concept used to decouple the team's development activity from downstream business planning. A PSI is a cohesive unit of functionality that **could** be delivered to the customer, but doing so is the decision of the product owner.
- There are seven activities:
 - The "sprint" is a defined time period, typically two to four weeks, in which the development team executes on an agreed scope
 - Backlog grooming is when the product backlog is examined and refined into increments that can be moved into the sprint backlog
 - Sprint Planning is where the scope is agreed
 - The Daily Scrum is traditionally held standing up, to maintain focus and ensure brevity
 - Sprint Execution is the development activity within the sprint
 - Sprint Review is the "public end of the sprint" when the stakeholders are invited to view the completed work
 - The Sprint Retrospective is held to identify lessons learned from the sprint and how to apply them in future work.

Scrum is well grounded in theory, although Scrum team members do not need to understand the theory to succeed with it. Like Lean UX, Scrum emphasizes high-bandwidth collaboration, dedicated multi-skilled teams, a product focus, and so forth.

Sidebar: Scrum and shu-ha-ri

In the Japanese martial art of aikido, there is the concept of shu-ha-ri, a form of learning progression.

- Shu: The student follows the rules of a given method precisely, without addition or alteration
- Ha: The student learns theory and principle of the technique
- Ri: The student creates own approaches and adapts technique to circumstance

Scrum at its most prescriptive can be seen as a shu-level practice; it gives detailed guidance that has been shown to work.

(See [\[Fowler2006\]](#) and [\[Cockburn2007\]](#), pp. 17-18.)

More on product team roles

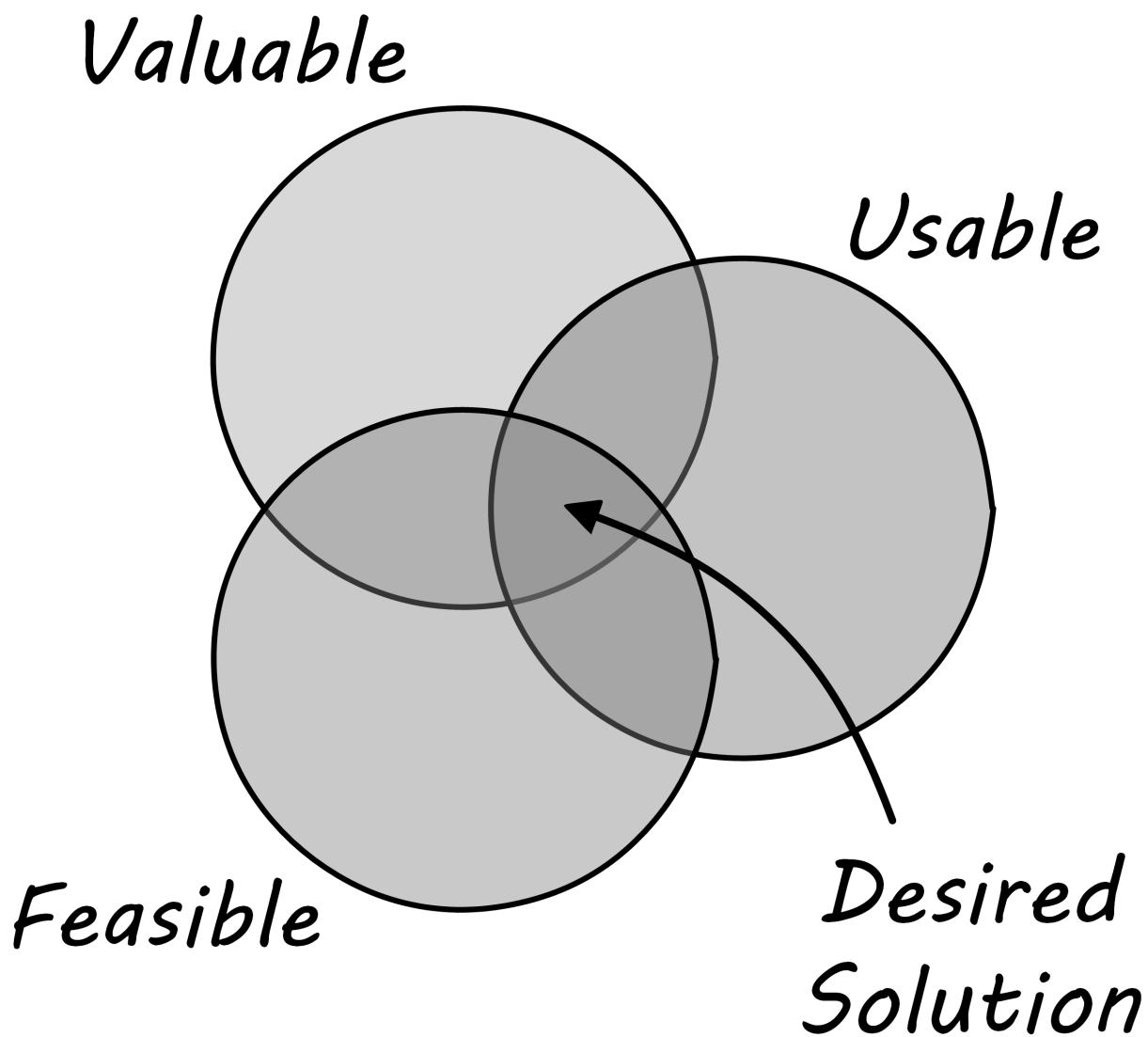
Boundaries are provided by the product owner and often come in the form of constraints, such as * I need it by June. * We need to reduce the per-unit cost by half. * It needs to run at twice the speed. * It can use only half the memory of the current version.

— Mike Cohn, Succeeding with Agile Software Development Using Scrum

Marty Cagan suggests that the product team has three primary concerns, requiring three critical roles [\[Cagan2008\]](#), (ch.12):

- Value: Product Owner/Manager
- Feasibility: Engineering
- Usability: User Experience Design

Jeff Patton represents these concepts as a Venn diagram:



similar to [Patton2014], ch.12

Finally, a word on the product manager. **Scrum** is prescriptive around the product **owner** role, but does not identify a role for product **manager**. This can lead to two people performing product management: a marketing-aligned "manager" responsible for high-level requirements, with the Scrum "product owner" attempting to translate them for the team. Marty Cagan warns against this approach, recommending instead that the product manager and owner be the same person, separate from marketing [Cagan2008], pp. 7-8.

In the next chapter, we will consider the challenge of product discovery—at a product level, what practices do we follow to generate the creative insights that will result in customer value?

Product discovery

What is a Product Manager? A Product Manager is the one person in the whole organization who owns the product Requirements effort. Requirements focuses on the WHAT, which means it isn't Development, which focuses on the HOW. And Marketing traditionally talks about the WHAT, but cannot necessarily decide what the WHAT should be. At least not at any useful level of detail. [\[Murphy2007\]](#)

— Jacques Murphy, Pragmatic Marketing

Now that we have discussed the overall concept of product management and why it is important, and how product teams are formed, we can turn more specifically to the topic of product discovery and design.

We have previously discussed the overall [digital business context](#), as a startup founder might think of the problem. But the process of discovery continues as the product idea is refined, new business problems are identified, and solutions (such as specific feature ideas) are designed and tested for outcomes.

NOTE In this book, we favor the idea that products are "discovered" as much or more than they are "designed." But you will see both terms used throughout this chapter. See the parable [the Flower and the Cog](#) for an illustration of the difference.

The presence of a section entitled "product discovery" in this book is a departure from other IT management textbooks. "Traditional" models of IT delivery focus on projects and deliverables, concepts we touched on in the [last chapter section](#) but that we will not explore in depth until later in the book.

However, the idea of "product discovery" **within** the large company is receiving more and more attention. Even large companies [benefit](#) when products are developed with tight-knit teams using fast feedback.

NOTE The term "intrapreneurship," credited to Gifford Pinchot, means "entrepreneurship inside a large company."

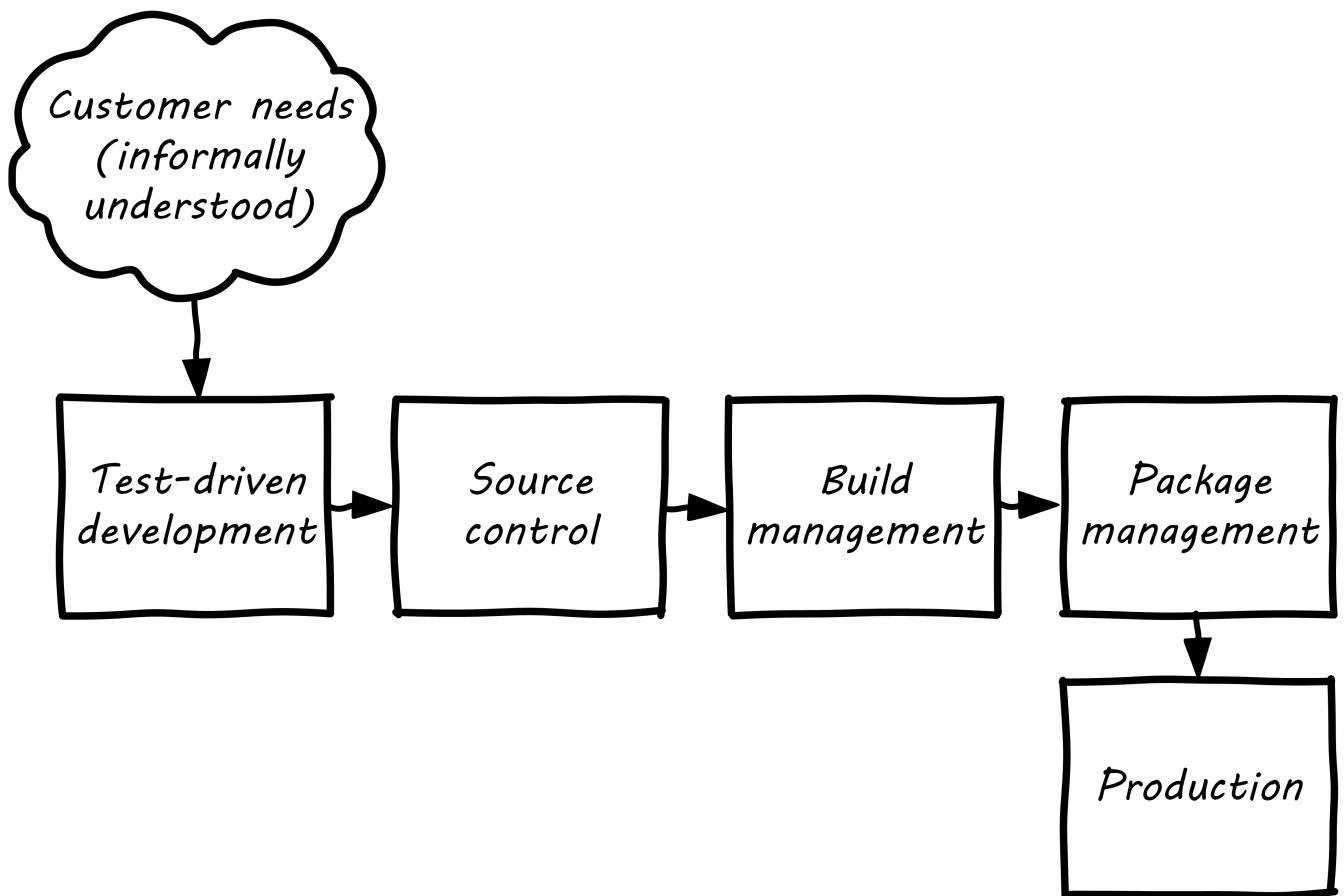
For our discussion here, the challenge with the ideas of projects and deliverables is that they represent approaches that are more [open loop](#), or at least delayed in [feedback](#). Design processes do not perform well when feedback is delayed. [System intent](#), captured as a user story or requirement, is only an hypothesis until tested via implementation and user confirmation.

Formalizing product discovery

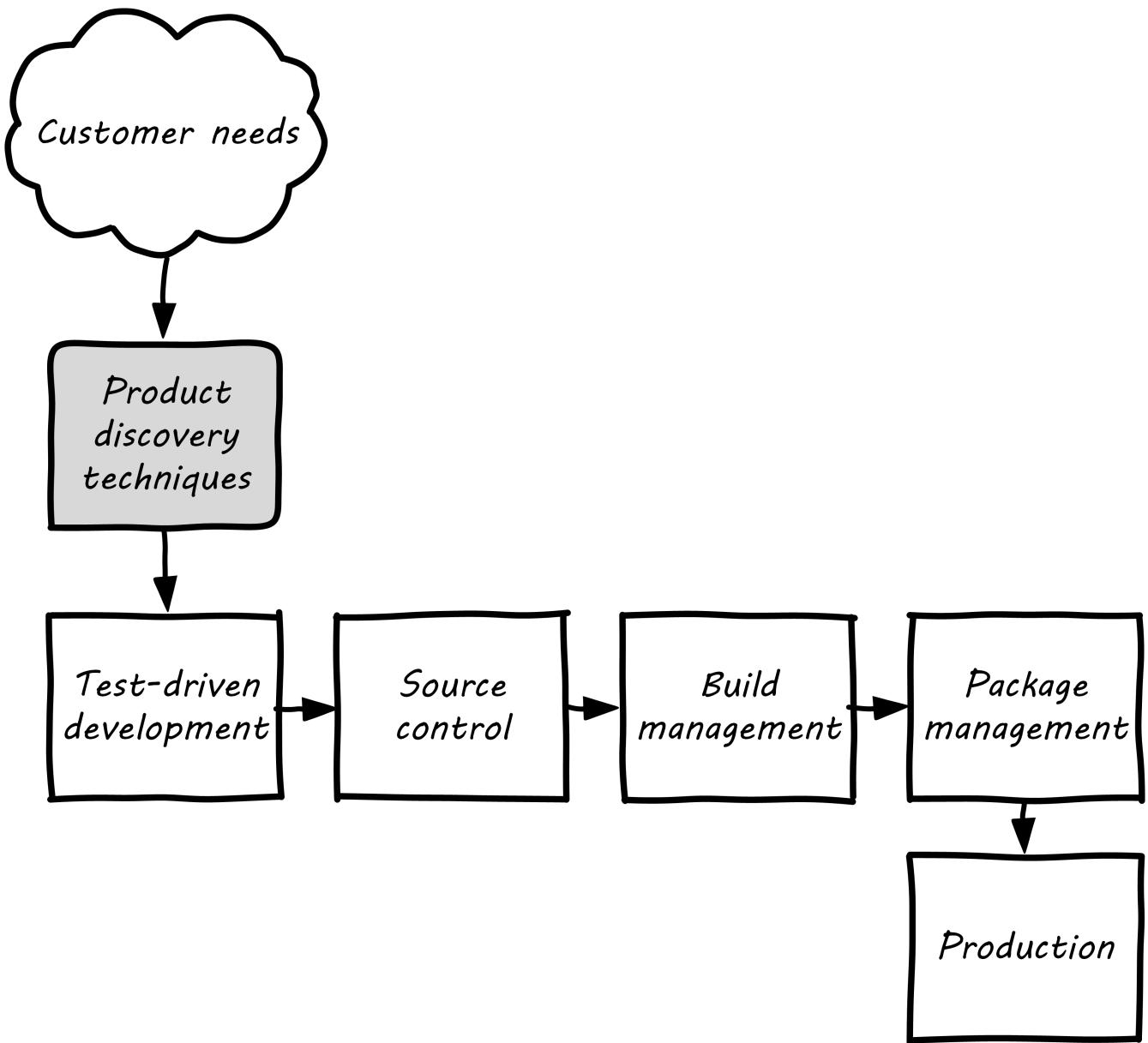
It is humbling to see how bad experts are at estimating the value of features (us included).

— Ronnie Kohavi, Online Experimentation at Microsoft

In Chapter 3, we needed to consider the means for describing [system intent](#). Even as a bare-bones startup, some formalization of this starts to emerge, at the very least in the form of test-driven development:



But, the assumption in our emergence model is that more formalized product management emerges with the formation of a team. As a team, we now need to expand "upstream" of the core delivery pipeline, so that we can collaborate and discover more effectively. Notice the grey box in the following:



The most important foundation for your newly formalized product discovery capability is that it must be **empirical and hypothesis-driven**. Too often, product strategy is based on the HIPPO: The Highest Paid Person's Opinion.



Beware of the HIPPO approach to product discovery [11: Image credit <https://www.flickr.com/photos/puliarfanita/6002022840>, downloaded 2016-09-22, commercial use permitted]

The problem with relying on "gut feel" or personal opinions is that people—regardless of experience or seniority—perform poorly in assessing the likely outcome of their product ideas.

Some well known research on this topic was conducted by Microsoft's Ronny Kohavi. In this research, Kohavi and team determined that "only about 1/3 of ideas improve the metrics they were designed to improve." [\[Kohavi2009\]](#) As background, the same report cites that:

- "Netflix considers 90% of what they try to be wrong"
- "75 percent of important business decisions and business improvement ideas either have no impact on performance or actually hurt performance" according to Qualpro (a consultancy specializing in controlled experiments)

It is therefore critical to establish a strong practice of data-driven experimentation when forming a product team, and avoid any cultural acceptance of "gut feel" or deferring to HIPPOs. This can be a difficult transition for the company founder, who has until now served as the de facto product manager.

Product discovery techniques

There are a wide variety of techniques and even "schools" of product discovery and design; we will consider a few representatives in this chapter section.

Of course, when you first started your journey in Chapter 1, you might also have used some of these techniques. But now that you are a team, you are formalizing and relying on these techniques.

These techniques are not mutually exclusive; they may be complementary. But at the more detailed,

digital product level, how do we develop hypotheses for testing, in terms of our products/services? We briefly mentioned User Story Mapping in our discussion of [system intent](#). But that is only one of many techniques. Other tools include:

- "Jobs to be done" analysis
- Impact mapping
- Business analysis & architecture

Job to Be Done

Customers don't want a quarter-inch drill. They want a quarter-inch hole.

— Theodore Levitt

If I'd asked the customer what they wanted, they would have said "faster horses."

— Henry Ford, (apocryphal)

The "Jobs to be Done" framework was created by noted Harvard professor Clayton Christensen, in part as a reaction against conventional [marketing](#) techniques that

"frame customers by attributes—using age ranges, race, marital status, and other categories that ultimately create products and entire categories too focused on what companies want to sell, rather than on what customers actually need." [\[Christensen2015\]](#)

"Some products are better defined by the job they do than the customers they serve," in other words. [\[Traynor2016\]](#) This is in contrast to many kinds of business and requirements analysis that focus on identifying different user personas (e.g. 45-55 married Black woman with children in the house). Jobs to be Done advocates argue that "The job, not the customer, is the fundamental unit of analysis" and that customer "hire" products to do a certain job. [\[Christensen2006\]](#)

To apply the Job to Be done approach, Des Traynor suggests filling in the blanks in the following [\[Traynor2016\]](#) :

Why do people hire your product?

People hire your product to do the job of ----- every ----- when -----. The other applicants for this job are -----, -----, and -----, but your product will always get the job because of -----.

Understanding the alternatives people have is key. It's possible that the job can be fulfilled in multiple different ways. For example, people may want certain software run. This job can be undertaken through owning a computer (e.g. having a data center). It can also be managed by hiring someone else's computer (e.g. using a Cloud provider). If one is not attentive and creative in thinking about the diverse ways jobs can be done, one is at risk for disruption.

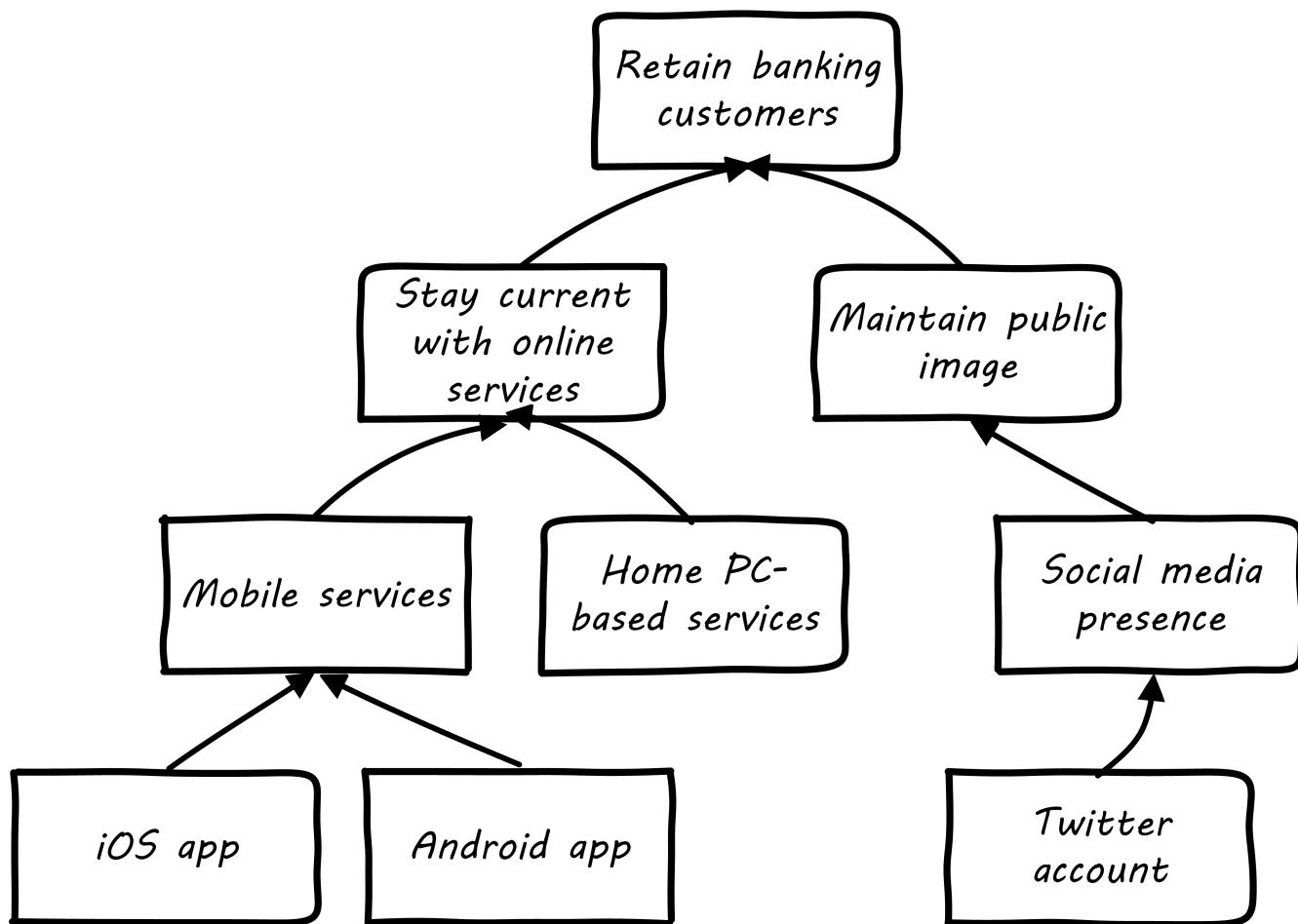
Impact mapping

Understanding the relationship of a given feature or component to business objectives is critical. Too often, technologists (e.g. software professionals) are accused of wanting "technology for technology's sake."

Showing the "line of sight" from technology to a business objective is therefore critical. Ideally, this starts by identifying the business objective. Gojko Adzic's *Impact Mapping: Making a big impact with software products and projects* [Adzic2012] describes a technique for doing so:

An *impact map* is a visualisation of scope and underlying assumptions, created collaboratively by senior technical and business people.

Starting with some general goal or hypothesis (e.g. generated through Lean Startup thinking), one builds a "map" of how the goal can be achieved, or hypothesis can be measured. A simple graphical approach can be used:



NOTE

Impact mapping is similar to mind mapping, and some drawing tools such as Microsoft Visio come with "Mind Mapping" templates.

The most important part of the impact map is to answer the question "Why are we doing this?" The impact map is intended to help keep the team focused on the most important objectives, and avoid less valuable activities and investments.

For example, in the above diagram, we see that a bank may have an overall business goal of customer retention. (It is much more expensive to gain a new customer, than to retain an existing

one, and retention is a metric carefully measured and tracked at the highest levels of the business.)

Through focus groups and surveys, the bank may determine that staying current with online services is important to retaining customers. Some of these services are accessed by home PCs, but increasingly customers want access via mobile devices.

These business drivers lead to the decision to invest in online banking applications for both the Apple and Android mobile platforms. This decision in turn will lead to further discovery, analysis, and design of the mobile applications.

The Business Analysis Body of Knowledge

One well-established method for product discovery is that of business analysis, formalized in the *Business Analysis Body of Knowledge* (BABOK), from the International Institute of Business Analysis [IIBA2015]

The Business Analysis Body of Knowledge (BABOK) defines business analysis as (p. 442):

The practice of enabling change in the context of an enterprise by defining needs and recommending solutions that deliver value to stakeholders.

BABOK is centrally concerned with the concept of requirements, and classifies them thus:

- Business requirements
- Stakeholder requirements
- Solution requirements
 - functional requirements
 - non-functional requirements
- Transition requirements

BABOK also provides a framework for understanding and managing the work of business analysts; in general it assumes that a BA capability will be established and that maturing such a capability is a desirable thing. This may run counter to the Scrum ideal of cross-functional, multi-skilled teams. Also as noted [above](#), the term "requirements" has fallen out of favor with some Agile thought leaders.

Discovery and design

Some of the most contentious discussions related to IT management and Agile come at the intersection of software and systems engineering, especially when large investments are at stake. We call this the "discovery versus design" problem.

Frequent criticisms of Lean Startup and its related digital practices are:

- They are relevant only for non-critical Internet-based products (e.g. Facebook and Netflix)

- Some IT products must fit much tighter specifications and do not have the freedom to "pivot" (e.g. control software written for aerospace & defense systems)

Parable: The flower and the cog

THUNK!

Hello. Where did you come from?

I fell. From that machine.

Machine?

Yes, that big loud thing that just passed by. And is now stopped over there.

Why is it stopped?

**Because I am no longer with it. The machine needs me to function. I am called a "cog."
Where did you come from?**

I am a flower. I grew from a seed.

You ... grew?

Yes.

You mean, no-one planned or designed you?

Not that I know of. What does it mean to be "designed" or "planned"?

I am part of a greater whole. The need for me was understood when that greater whole was conceived. I was designed to fit a very particular place.

They had to try making me out of different metals, and different ways to make me. This took some time and effort—longer than was planned, in fact. But it was always understood that there would need to be a cog in a certain place in the machine.

Interesting. So you will never be more than you are?

No. I will always be a cog. They might make a different machine, with different cogs, but they will not be me. Are you part of a machine?

No. I grew here because it suited me. I have continued to grow for a couple years. Eventually I may grow 20 feet tall, if the conditions remain good. I can adapt to other plants, and find my way around them to the sunlight and the water I need. Or I may stay smaller, if I can't get the sunlight I need. Or I may die.

Aren't you part of a system that defines your purpose?

I don't know. Sometimes I think I am a system myself, made up of my roots, stem, leaves, and

flower. There are insects living on me who rely on me for food and shelter. And I have the freedom to grow into one of the largest trees in this area. That is worth it to me.

Interesting. Well, it is good you are growing where you are, and not twenty feet further in that direction.

Why?

Because when they find me, or replace me and fix the machine, it will continue to clear all the land over there.

Oh.

VOICES: "Hey Joe, here's that gear the tractor must have thrown."

"Good, grab it and I'll see if I can't get it back in place at least temporarily until we can figure out why it happened."

Bye.

Goodbye. Nice talking to you. Good luck.

Thanks. You too.

The above parable is meant to illustrate two very different product development worlds. Some product development is constrained by the overall system it takes place within. Other product development has more freedom to grow in different directions—to "discover" the customer.

The cog represents the world of classic systems engineering—a larger objective frames the effort, and the component occupies a certain defined place within it. And yet, it may still be challenging to design and build the component, which can be understood as a product in and of itself. Fast feedback is still required for the design and development process, even when the product is only a small component with a very specific set of requirements.

The flower represents the market-facing digital product that may "pivot," grow and adapt according to conditions. It also is constrained, by available space and energy, but within certain boundaries has greater adaptability.

Neither is better than the other, but they do require different approaches. In general, we are coming from a world that saw digital systems strictly as cogs, and moving towards a world in which digital systems are more flexible, dynamic, and adaptable.

And, when digital components have very well understood requirements, usually we purchase them from specialist providers (increasingly "as a service"). This results in increasing attention to the "flowers" of digital product design, as acquiring the "cogs" is relatively straightforward (more on this in the chapter 8 section on sourcing.).

Design



[12: Image credit <https://www.flickr.com/photos/djs1021/101948321/>, downloaded 2016-09-19, commercial use permitted]

Everyone designs who devises courses of action aimed at changing existing situations into preferred ones [Simon1988].

— Herbert Simon

The art of making useful things beautiful and beautiful things useful.

— unknown

Once we have discovered at least a direction for the product's value proposition, and have started to understand the functions it must perform, we begin the activity of design.

Design, like most other topics in this survey text, is a broad and complex area with varying definitions and schools of thought. The Herbert Simon quote at the beginning of this section is frequently cited.

Design is an ongoing theme throughout the humanities, encountered in architecture (the non-IT variety), art, graphics, fashion, and commerce. It can be narrowly focused, such as the question of what color scheme to use on an app or web page. Or it can be much more expansive, as suggested by the field of design thinking. We'll start from the expansive vision and drill down into a few interesting topics.

Design thinking

Design thinking is essentially a human-centered innovation process that emphasizes observation, collaboration, fast learning, visualization of ideas, rapid concept prototyping, and concurrent business analysis, which ultimately influences innovation and business strategy. [\[Lockwood2009\]](#)

— Thomas Lockwood, Design Thinking

Design thinking is a recent trend with various definitions, but in general combines a design sensibility with problem solving at significant scale. It usually is understood to include a significant component of systems thinking. As Tom Fisher, author of *Designing Our Way to a Better World* [\[Fisher2016\]](#), notes:

We've been doing a lot of work in this area of "design thinking," which takes the thought process and the methods that have been developed for millennia around the design of physical things — products, buildings, cities — and applies that to the so-called invisible world of design, which is all of the systems and organizations that are designed, but we don't think of them as being designed. And we're seeing a lot of these systems not working very well. [\[Peck2016\]](#)

Design thinking is the logical evolution of disciplines such as user interface design, when such designs encounter constraints and issues beyond their usual span of concern. Although it has been influential on Lean UX and related works, it is not an explicitly digital discipline.

There are many design failures in digital product delivery. What is often overlooked is that the entire customer experience of the product is a form of design.

Consider for example Apple Computer. Their products are admired worldwide and cited as examples of "good design." Often, however, this is only understood in terms of the physical product, for example an iPhone or a MacBook Air. But there is more to the experience. Suppose you have technical difficulties with your iPhone, or you just want to get more value out of it. Apple created its popular Genius Bar support service, where you can get support and instruction in using the technology:

Notice that the product you are using is no longer just the phone, or computer. **It is the combination of the device PLUS your support experience.** This is essential to understanding the modern practices of design thinking and Lean UX. As Jeff Sussna, author of *Designing Delivery*, notes, "In order to provide high-quality, digitally infused service, the entire delivery organization must function as an integrated whole." [\[Sussna2015\]](#), p. 18.

Hypothesis testing

The concept of hypothesis testing is key to product discovery and design. The power of scalable Cloud architectures and fast Continuous Delivery pipelines has made it possible to test product hypotheses against real-world customers at scale and in real time. Companies like Netflix and Facebook have pioneered techniques like "canary deployments" and "A/B testing."

In these approaches, two different feature approaches are tried out simultaneously and the business results are measured. For example, are customers more likely to click on a green button or a yellow one? Testing such questions in the era of packaged software would have required lab-based usability engineering approaches, which risked being invalid because of their small sample size. Testing against larger numbers is possible, now that software is increasingly delivered as a service.

Usability and interaction

At a lower level than the holistic concerns of design thinking, we have practices such as usability engineering. These take many forms. There are any number of systematic and well-researched approaches to

- usability, interaction design ([\[Cooper2009\]](#), [\[Isaacs2002\]](#), [\[Tidwell2006\]](#), [\[Bank2016\]](#))
- visualization ([\[Card1999\]](#), [\[Tufte2001\]](#))

and related topics.

All such approaches, however, should be used in the overall Lean Startup/Lean UX framework of hypothesis generation and testing.

Features vs. components

As your product evolves, you may see an increasing differentiation of concerns, between features and components.

Features are **what** your product **does**. They are what the customers perceive as valuable. "Scope as viewed by the customer" as [\[Kennaley2010\]](#) says (p. 169).

Components are **how** your product is **built**, such as database vs. Web components. In other words, they are a form of infrastructure (but infrastructure you may need to build yourself, rather than just spin up in the Cloud).

Feature teams are dedicated to a clearly defined functional scope, while component teams are defined by their technology platform. Component teams may become shared services, which need

to be carefully understood and managed (more on this in Section 3).

It may be easy to say that features are more important than components, but this can easily be carried too far. Do you want each feature team choosing its own database? Not recommended. Allowing feature teams to define their own technical direction can result in brittle, fragmented architectures, technical debt, and rework.

Software product management needs to be a careful balance between these two perspectives. Some [recommend](#) a ratio of:

- 70-80% Feature teams to
- 20-30% Component teams

We will be revisiting this more systematically in Section III.

Assorted topics in Product Management

Collaborative

Status: Like most topics sections, work in progress that will probably not be completed until 2nd or later drafts.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

===== The production spectrum
* Make to Stock
* Assemble to Order
* Make to Order
* Engineer to Order

[more]

===== the "efficiency" fallacy
Managed services are not 3 truck rolls: The screwdriver truck, the wire cutters truck, the conduit truck... efficiency

Data-driven product discovery (foreshadow ch 11)

===== misc

Poppendieck wastes as cited by Cobb, p177

<https://medium.com/@cwodtke/the-myths-of-ux-design-product-design-whatever-they-call-it-this-week-ef37a39cac6b#.7uj1khbdh> - product mgmt, design, engineering - good breakdown

<https://twitter.com/johncutlefish/status/754389401728188416>

unused software: information generation (Standish revised understanding)

what makes a good business analyst?

Lean Enterprise "Turn central IT into a product development organization."

#include::book/2-section-II/2.04-chap-4/2.04.10-domain-model.adoc[]

Conclusion

As with most other chapters in this book, this brief survey cannot do justice to such a broad field.

Product-centric thinking is at the core of delivering digital value. Without a clear understanding of the product and its context, investments of time and resources are at risk.

Product management requires close collaboration between individuals of different skills. Team structures, practices, and culture are critical factors for success. Establishing an experimental mindset is essential; research shows that product ideas must be validated as expert opinion is of minimal value.

Product discovery and design is a broad field; design thinking is an important influence, as design spans more than just the user interface.

Discussion questions

- Can you think of examples from your own experience where you or someone else confused "output" (e.g. deliverable) with "outcome"?
- What is your experience with being on teams? Have you ever been on a team where people felt the **psychological safety** to take risks? Or the opposite?
- Can you identify two or three distinct features (as defined **above**) for a commercial product

such as Facebook or Netflix? Try to "reverse engineer" features that might have been individual areas of investment.

- Can you think of any components (as defined [above](#)) you will need?

Research & practice

- If you have team product ideas from previous discussions, consider them in terms of the [Cagan/Patton](#) questions and prepare an analysis:
 - Is it valuable?
 - Is it usable?
 - Is it feasible?
- Identify and analyze a situation where "design" requires a broad, design thinking understanding of both physical product as well as the service interactions surrounding it. Present to your class.
- Review the Scaled Agile Framework's descriptions of [features](#) and [user stories](#). Document one feature consisting of several stories for your product.
- Research Behavior-Driven Development and consider its potential value for you.

Further reading

Books

- Gojko Adzic, [Impact Mapping: Making a Big Impact with Software Products and Projects](#)
- Chris Bank and Jerry Cao, [The Guide to Usability Testing](#)
- Marty Cagan, [Inspired: How To Create Products Customers Love](#)
- Tom Fisher, [Designing Our Way to a Better World](#)
- Jeff Gothelf, [Lean UX: Applying Lean Principles to Improve User Experience](#)
- Thomas Lockwood, [Design Thinking: Integrating Innovation, Customer Experience, and Brand Value](#)
- Jeff Patton, [User Story Mapping: Discover the Whole Story, Build the Right Product](#)
- Eric Ries, [The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses](#)
- Ken Rubin, [Essential Scrum: A Practical Guide to the Most Popular Agile Process](#)
- Chris Sims and Hilary Louise Johnson, [Scrum: a Breathtakingly Brief and Agile Introduction](#)
- Jeff Sussna, [Designing Delivery: Rethinking IT in the Digital Service Economy](#)

Articles

- [Online Experimentation at Microsoft](#)
- [The Myths of UX Design/ Product Design/Whatever They Call It This Week](#)
- [Building a design-driven culture](#)

- Your ultimate guide to Minimum Viable Product
- Feature team: beyond the buzzword
- SAFe on Features and Components
- Lean Startup: The Right Strategy or Fool's Gold?
- Where Should Product Management Report?
- 8 Things To Use in “Jobs-To-Be-Done” Framework For Product Development
- The past and future of product management
- Creating Product Management at a Startup

Professional

- Product Development and Marketing Association
- Scrum Alliance

second draft

<https://www.amazon.com/Inspired-Create-Products-Customers-Love/dp/0981690408>

<https://www.intercom.io/books/product-management>

review following deck for inclusions here

* <http://www.slideshare.net/Innolution/agile-is-riskmanagementkenrubininnolution>

Chapter 5: Work Management

Introduction to Chapter 5

NOTE

"Work management" is an unusual title, deliberately chosen to reflect earlier stages of organizational growth. At this point, neither formal project management, nor a fully realized process framework are needed, and the organization may not see a need to distinguish precisely between types of work processes. "It's all just work" at this stage.

As two more assistants were added, however, coordination problems did arise. One day, Miss Bisque tripped over a pail of glaze and broke five pots; another day, Ms. Raku opened the kiln to find that the hanging planters had all been glazed fuchsia by mistake. At this point, she realized that seven people in a small pottery studio could not coordinate all their work through the simple mechanism of informal communication.

— Henry Mintzberg, *Structure in Fives*

- What do we need to do?
- In what order?
- Who is doing it?
- Do they need help?
- Do they need direction?
- When will they be done?
- What do we mean by done?

These are fundamental questions for any business larger than a couple co-founders.

You're now a team. You need to get work done. When your startup hired its first employees, and you went from two people to three or four, you almost certainly had to think about this. You are now getting feedback from both your users and product leadership calling for prioritization, the allocation of resources, and the tracking of effort and completion.

People have different responsibilities and specialties, yet there is a common vision for delivering an IT-based product of some value. How do you keep track of the work? Mostly, you are in the same location, but people sometimes are offsite or keeping different hours. Beyond your product strategy, you have support calls increasingly coming in that result in fixes and new features, and it's all getting complicated.

You already have a product owner. You now institute Scrum practices of a managed backlog, daily standups and sprints. You also use Kanban-style task boards, which are essential for things like support-driven work, but the relationship of these to your Scrum practices is a matter of some ongoing debate. You don't think you need full-blown project management — yet.

You also may become aware of the idea of "ticketing," if you have people with previous service desk experience. How this relates to your Scrum/Kanban approach is also a question.

Furthermore, while we covered Agile principles and practices in some detail in chapters 3 and 4, we did not discuss **why** they work. In this chapter, we will cover the Lean theory of product management that provides a basis for Agile practices, in particular the work of Don Reinertsen.

Chapter 5 outline

- Why work management?
- Kanban

- Origins
- Practices
- Relationship of Scrum to Kanban
- The shared mental model
- Time and space shifting
- The help desk and the "ticket"
- Lean product management
- Advanced topics
 - Demand/supply/execute
 - cadence and synch
 - velocity and flow

Chapter 5 learning objectives

- Understand the basic concept of task management
- Define Kanban and identify its key practices
- Compare and contrast Scrum and Kanban
- Understand concepts from Lean Product Management

Why work management?

Task management is the process of managing a task through its life cycle. It involves planning, testing, tracking and reporting. Task management can help either individuals achieve goals, or groups of individuals collaborate and share knowledge for the accomplishment of collective goal

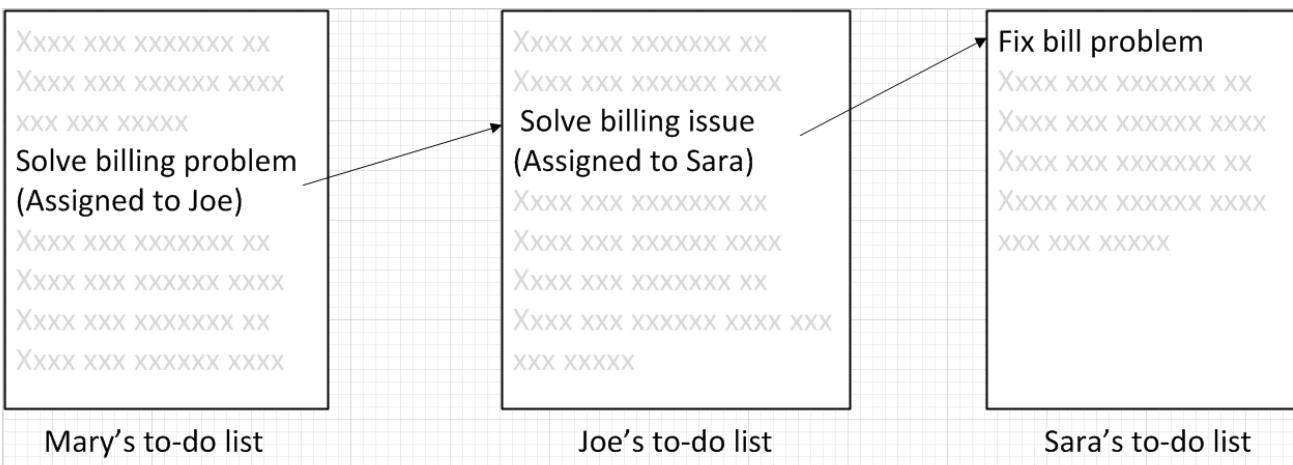
— Wikipedia, https://en.wikipedia.org/wiki/Task_management

Work management may start with verbal requests, emails, even postal mail. The design process may result in a wide variety of deliverables that also initiate work activity. As your product matures, the product itself generates both routine and non-routine work. Some of the work depends on other work getting done. Sometimes you do not realize this immediately. All of this needs to be tracked.

If you ask your colleague to do one thing, and she doesn't have anything else to do, it's likely that the two of you will remember. If you ask her to do four things over a few days, you **might** both still remember. But if you are asking for new things every day, it is likely that some things will get missed.

You each might start keeping your own “to do” list, and this mechanism can handle a dozen or two dozen tasks.

Consider an example of three people, each with their own to do list:



In this situation, each person has their own “mental model” of what needs to be done, and their own tracking mechanism. We don’t know how the work is being transmitted: emails, phone calls, hallway conversations (“Say Sara, there is an issue with Customer X’s bill, can you please look into it?”)



Figure 1. Task handoffs present risk [14: Image credit

<https://www.flickr.com/photos/chrisbrooks/42588998>, downloaded 2016-10-07, commercial use permitted]

But what happens when there are three of you? You ask coworker B to do something and in order to complete it she needs something from coworker C, whom you are also asking to complete tasks. As an organization scales, this can easily lead to confusion and “dropped balls.”

At some point, you need to formalize your model of the work, how it is described, and how it flows. This is such a fundamental problem in human society that many different systems, tools, and processes have been developed over the centuries to address it.

Probably the most important is the shared task reference point. What does this mean? The “task” is

reified (see definition) into a common artifact.

For example, a “ticket” may be created. Or a “story.” At our current level of understanding, there is little difference between these concepts. The important thing they have in common is an independent existence. That is, Mary, Joe, and Sara might all change jobs, but the artifact persists independently of them.

Notice also that the artifact - the ticket, the post-it note - is **not** the actual task, which is an intangible, consensus concept. It is a *representation* of this intangible “intent to perform.” We will discuss these issues of representation further in Chapter 11.

A complex IT-based system is not needed, if you are all in the same room! (Nor for that matter a complex process framework, such as [ITIL](#). There is risk in using such frameworks at this stage of evolution - they add too much overhead for your level of growth.)

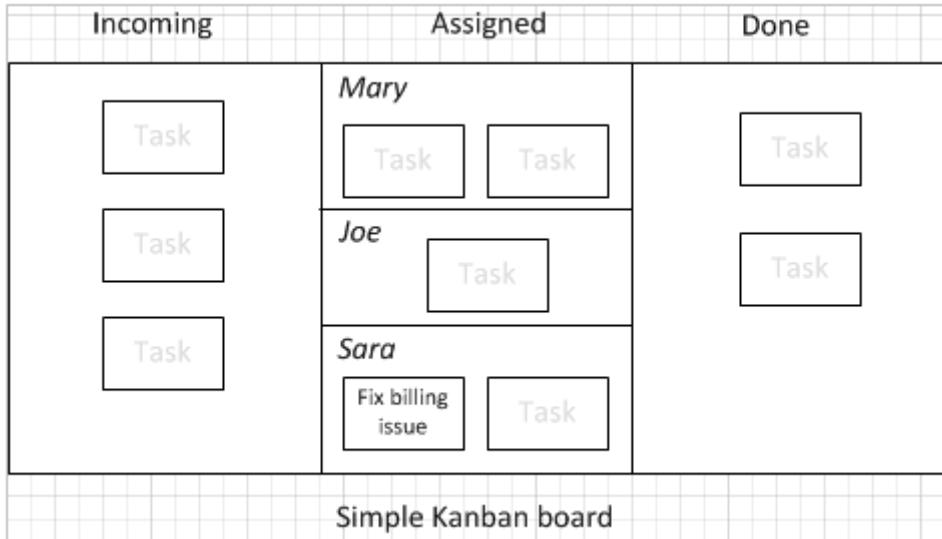
It’s also still too early for formal project management. The “project team” would be most or all of the organization, so what would be the point?

A shared white board in a public location might be all that is needed. This gives the team a “shared mental model” of who is doing what.

Task	Assigned to
Xxxx XXX XXXXXXXX XX	Xxxx
Xxxx XXX XXXXXXX XXXX	Xxxx
XXX XXX XXXXXX	Xxxxxx
Solve billing problem	Sara
Xxxx XXX XXXXXXXX XX	Xxxx
Xxxx XXX XXXXXXX XXXX	Xxxx
Xxxx XXX XXXXXXXX XX	Xxxxxx
Xxxx XXX XXXXXXX XXXX	Xxxxxx

Common to-do list

The design of the task board above has some issues, however. After the team gets tired of erasing and rewriting the tasks and their current assignments, they might adopt something more like this:



The rectangles labeled “Task” are index cards or sticky notes. The board itself might be a white board, or a cork bulletin board with push pins. It doesn’t really matter. The important thing is that, at a glance, the entire team can see its flow of work and who is doing what.

This is usually called a “Kanban board” (from [\[Anderson2010\]](#)), although Scrum originator Sutherland calls it a “Scrum board”. The board at its most basic is not specific to either methodology. (See the sidebar on Scrum and Kanban.)

The basic board is widely used because it is a powerful artifact. Behind its deceptive simplicity is considerable industrial experience and relevant theory from operations management and human factors [\[Anderson2010\]](#). However, it has scalability limitations. What if the team is not all in the same room? We will cover this in Section 3.

Queues and limiting work in process

Queues matter because they are economically important, they are poorly managed, and they have the potential to be much better managed. Queues profoundly affect the economics of product development. They cause valuable work products to sit idle, waiting to access busy resources. This idle time increases inventory, which is the root cause of many other economic problems...queues hurt cycle time, quality, and efficiency.

— Don Reinertsen, Principles of Product Development Flow



Figure 2. A queue [1: Image credit <https://www.flickr.com/photos/hktang/4243300265>, downloaded 2016-10-07, commercial use permitted]

Even at this stage of our evolution, with just one co-located collaborative team, it's important to consider work in progress and how to limit it. One topic we will emphasize throughout the rest of this book is *queuing*.

What is a queue? A queue, intuitively, is a collection of tasks to be done, being serviced by some worker or resource in some sequence.

IMPORTANT

Queuing theory is an important branch of mathematics used extensively in computing, operations research, networking, and other fields. It's a topic getting much attention of late in the Agile and related movements, especially as it relates to team productivity.

Lean manufacturing has been an important movement throughout the 20th century, and one of its central principles is to limit work in process. Work in process is obvious on a shop floor, because physical raw materials (inventory) are quite visible.



Figure 3. Physical Work in Process [2: Image credit <https://www.flickr.com/photos/jayburkhalter/6679328027/>, downloaded 2016-10-07, commercial use permitted]

Don Reinertsen, quoted above, developed the insight that product design and development had an **invisible** inventory of "Work in Process" that he called Design in Process (DIP). Just as managing physical work in process on the factory floor is key to a factory's success, so correctly understanding and managing design in process is essential to all kinds of research and development organizations—**including digital product development**. In fact, because digital

systems are largely invisible even when finished, understanding their work in process is even more challenging.

It is easy and tempting for a product development team to accumulate excessive amounts of work in process. And, to some degree, having a rich backlog of ideas is an asset. But, just as some inventory (e.g. groceries) is perishable, so are design ideas. They have a limited time in which they might be relevant to a customer or a market. Therefore, accumulating too many of them at any point in time can be wasteful.

What does this have to do with queuing? Design in process is one form of queue seen in the digital organization. Other forms include unplanned work (incidents and defects), implementation work, and many other concepts we'll discuss in this chapter.

But at the end of the day, whether it is a "Requirement," a "User Story," an "Epic," "Defect," "Issue," or "Service Request," it's **all just work**. It needs to be logged, prioritized, assigned, and tracked to completion. Queues are the fundamental concept for doing this, and it's critical that digital management specialists understand this.

We will revisit the topic of queuing in Chapter 9. For now, some rules of thumb:

- Finish what you start, if you can, before starting anything else. When you work on three things at once, the multi-tasking wastes time and it takes you three times longer to get any one of the things done. (More on [multi-tasking](#) in this chapter.)
- Infinitely long to-do lists (backlog) sap motivation. Consider limiting backlog as well as work in process.
- Visibility into work in progress is important for the collective mental model of the team.

Introducing Kanban

To understand Kanban it is advisable to discuss Lean briefly.

Kanban is a term originating from Lean and the Toyota Production System. Originally, it signified a “pull” technique in which materials would only be transferred to a given workstation on a definite signal that the workstation required the materials. This was in contrast to “push” approaches where work was allowed to accumulate on the shop floor, on the (now discredited) idea that it was more “efficient” to operate workstations at maximum capacity.

There is a tremendous wealth of material available on Lean history and theory, and the IT student is urged to become familiar with it.

IMPORTANT

Historically, there has been resistance in IT circles to learning lessons derived from manufacturing, due to misperceptions that manufacturing is too “repetitive” compared to IT systems development.

This resistance is disappearing with the new generation of thought leadership centered on Agile methods and related themes such as DevOps.

David J. Anderson was a product manager at Microsoft who was seeking a more effective approach to managing software development. In consultation with Don Reinertsen (introduced in the previous chapter) he applied the original concept of Kanban to his development activities.[\[Anderson2010\]](#)

Unlike Scrum, with its defined, high-pressure sprints, Kanban is a continuous process with no specified cadence. Work is “pulled” from backlog into active attention as resources are freed from previous work.

This is perhaps the most important aspect of Kanban - the idea that work is not accepted until there is capacity to perform it.

You may have a white board covered with sticky notes, but if they are stacked on top of each other with no concern for worker availability, you are not doing Kanban. You are accepting too much work in process and you are likely to encounter a "high-queue state" in which work becomes slower and slower to get done.

There are deeper philosophical and cultural tenets embedded in Kanban beyond workflow. Anderson and his colleagues continue to evolve Kanban into a more ambitious framework. Mike Burrows [\[Burrows2014\]](#) identifies the following key principles

- Start with what you do now.
- Agree to pursue evolutionary change.
- Initially, respect current processes, roles, responsibilities, and job titles.
- Encourage acts of leadership at every level in your organization —from individual contributor to senior management.
- Visualize.
- Limit Work-in-Progress (WIP).
- Manage flow.
- Make policies explicit.
- Implement feedback loops.
- Improve collaboratively, evolve experimentally (using models and the scientific method).

[Reinertsen 204-205 on kanban]

Scrum, Kanban, or both?

So, do you choose Scrum, Kanban, both, or neither?

We can see in comparing Scrum and Kanban that their areas of focus are somewhat different.

- Scrum has achieved widespread success and formalization, which is why Scrum training is widespread and generally consistent in content.
- Kanban is seen as more flexible but this comes at a cost of more management overhead. It

requires more interpretation to translate to a given organization's culture and practices.

- “Scrum is not well suited to highly interrupt-driven work.” [\[Rubin2012\]](#). (But if you are too small, it may be difficult to segregate out interrupt-driven work! We will discuss the issues around interrupt-driven work further in Chapter 06.)
- Finally, hybrids exist (Ladas' “Scrumban,” [\[Ladas2009\]](#)).

More on this in Section III.

Sidebar: Kanban, process, and respect for people

Processes cross functions. Processes (e.g. Quote to Cash, Idea to Production) are what actually deliver value. We need both process and function; we can't organize solely around either. Therefore, get used to matrix management.

The above statements are the management dogma I grew up with. Considered to be innovative at the time, originated by the likes of Michael Hammer and (more rigorously) Geary Rummler. (See Paul Harmon's [excellent obituary of Geary Rummler](#).)

But consider a day in the life in the modern corporation: dozens or hundreds of cross functional processes that have become a daily barrage of work on the people housed in the functional areas. Driven by back end automation, front ended by automated (or even manual) email notifications, these required activities include:

- service requests,
- compliance certifications,
- asset validations,
- provisioning requests,
- capacity assessments,
- change approvals,
- training obligations,
- performance assessments,
- audit responses,
- expense reporting,
- travel approvals,

and more - infinitely more. This is the modern, “process”-centric enterprise.

True, any one of these processes is (often) a relatively small burden to perform. Taken together, however, it's the nightmare of [Adminisdribble](#): The daily death by a thousand cuts of required, cross-functional process.

Continual context switching. End to end flow in concept across functions, but little experienced by the multitasking individual. Invisible, unmanaged demand. A thousand channels for flow, but no throughput, because of gridlock.

What drives this? The combination of permanent downsizing, continually increasing governance & compliance burdens, and the assumption that the enterprise can always tolerate another process. Overall organizational capacity is rarely considered, especially for "supporting" processes. And there is no appreciation for the emergent behavior when these myriads of processes interact in unexpected ways.

This is why kanban is so radical. Within a limited scope, it is starting to challenge the IT management dogma at least. User stories, incidents, defects, service requests, changes... they are all "just work." Just another sticky note on the board. Someone has to perform the work; someone has to allocate time and attention to it. A simple but utterly profound change in orientation. Kanban is the labor movement of the 21st century.

It seems to me that true "respect for people" (as advocated by Lean) of course must start with the recognition that work takes people's time. I think the modern, process centric enterprise has lost sight of that reality. Is Kanban the answer?

originally posted at <http://www.lean4it.com/2014/08/kanban-vs-process.html>

The shared mental model of the work to be done

Joint activity depends on interpredictability of the participants' attitudes and actions. Such interpredictability is based on common ground — pertinent knowledge, beliefs and assumptions that are shared among the involved parties. [Klein2005]

— Gary Klein et al, Common Ground and Coordination in Joint Activity

The above quote reflects one of the most foundations of team collaboration: a common ground, a base of "knowledge, beliefs, and assumptions" enabling collaboration and coordination. Common ground is an essential quality of successful teamwork, and we will revisit it throughout the book. There are many ways in which common ground is important, and we will discuss some of the deeper aspects in terms of information in chap-ent-info-mgmt[chapter 11]. However, starting with the "definition of done" is a good place to introduce the idea of common ground.

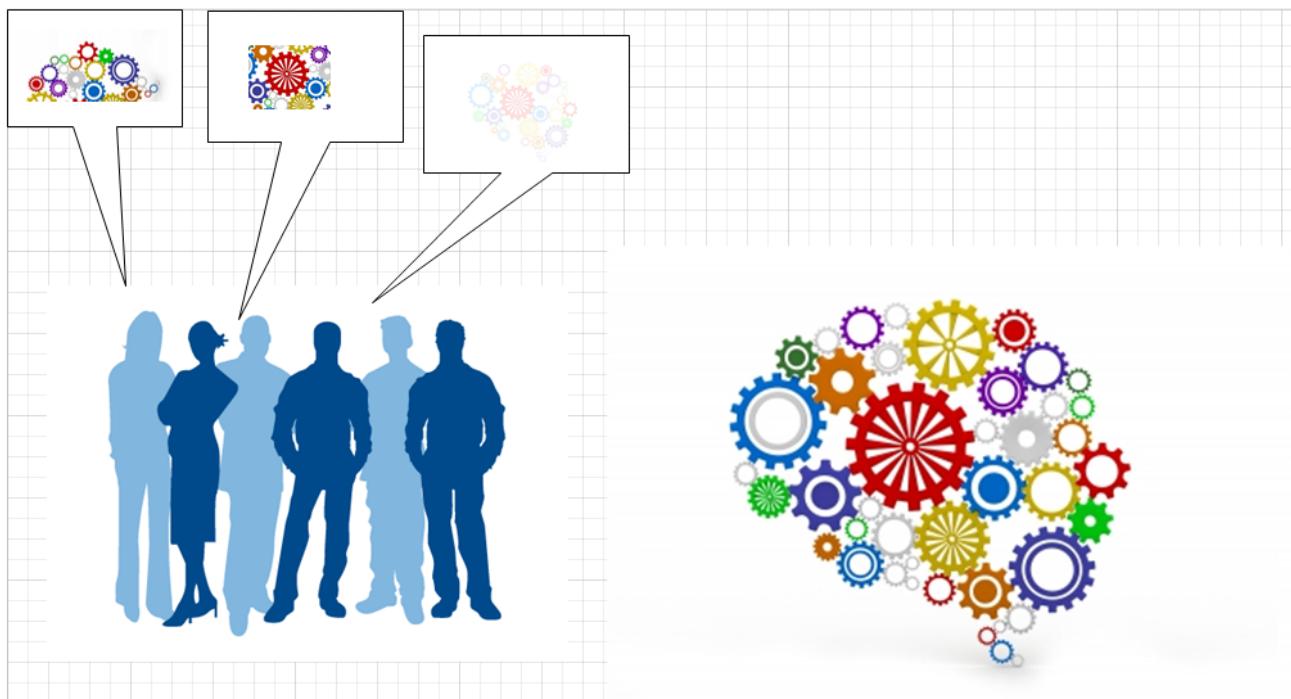
As work flows through the system performing it, understanding its status is key to managing it. One of the most important mechanisms for doing this is to simply define what is meant by "done." The Agile Alliance [states](#):

"The team agrees on, and displays prominently somewhere in the team room, a list of criteria which must be met before a product increment, often a user story, is considered "done". Failure to meet these criteria at the end of a sprint normally implies that the work should not be counted toward that sprint's velocity."

This idea of defining "done" can be extended by the team to other concepts such as "blocked." The important thing is that this is all part of the team's shared mental model, and is best defined by the team and its customers. (However, governance and consistency concerns may arise if teams are too

diverse in such definitions.)

Whether you choose Scrum, Kanban, or choose not to label your work management at all, the important thing is that you are creating a shared mental model of the work: its envisioned form and content, and your progress towards it.



Andon, and the Andon Cord

Another related concept from Lean is the Japanese idea of andon:

Andon is a manufacturing term referring to a system to notify management, maintenance, and other workers of a quality or process problem. The centrepiece is a signboard incorporating signal lights to indicate which workstation has the problem. The alert can be activated manually by a worker using a pullcord or button, or may be activated automatically by the production equipment itself. The system may include a means to stop production so the issue can be corrected. Some modern alert systems incorporate audio alarms, text, or other displays.

— Wikipedia, [https://en.wikipedia.org/wiki/Andon_\(manufacturing\)](https://en.wikipedia.org/wiki/Andon_(manufacturing))

The Andon Cord is a well known concept in Lean Manufacturing. It originated with Toyota, where line workers were empowered to stop the production line if any defective materials or assemblies were encountered. Instead of attempting to work with the defective input, the entire line would shut down and all concerned would establish what had happened and how to prevent it. The concept of andon cord concisely summarizes the Lean philosophy of employee responsibility for quality at all levels. [Ohno1988]

Stopping the machine when there is trouble forces awareness on everyone. When the problem is clearly understood, improvement is possible. Expanding this thought, we establish a rule that even in a manually operated production line, the workers themselves should push the stop button to halt production if any abnormality appears.

— Taiichi Ohno

This idea of a shared reference point, informing the shared mental model of the team, is essential.

This is why a shared kanban board, with its dimensionality, is more informative than a simple to-do list (e.g. in a spreadsheet). The cards occupy two dimensional space and are moved over time to signify activity, both powerful cues to the human visual processing system.

Visualization

[Information radiators]

Why are shared visual representations important? Depending on how you measure, between 40% to as much as 80% of the human cortex is devoted to visual processing. Furthermore, visual processing [dominates mental activity](#), consuming more neurons than the other four senses combined [\[Sells1957\]](#). Visual representations are powerful communication mechanisms, well suited to our cognitive abilities.

This is why Kanban boards located in the same room are so prevalent. They communicate and sustain the shared mental model of a human team. Similarly, monitoring tools for systems operation make use of various visual clues. Human visual orientation is also why Enterprise Architecture persists. People will always draw to communicate. (More on EA in Chapter 10.)

Definition of done

[Shared mental model, consensus alignment]

Thoughtworks writeup - BA has to agree with dev
<https://t.co/2X92CFcKqU>

Importance of code reviews

Time and space shifting

At some point your team will be faced with the problems of time and/or space shifting. People will be on different schedules, or in different locations, or both. There are two things we know about this. First, it leads to sub-optimal team communications and performance. Second, it is inevitable.

The need for time and space shifting is one of the major drivers for more formalized IT systems. It is difficult to effectively use a physical Kanban board if people aren't in the office. The outcome of the daily standup needs to be captured for the benefit of those who could not be there.

However, acceptance of time and space shifting may lead to more of it, even when it is not absolutely required. A constant pressure and questioning is recommended, given the superior bandwidth of face to face communication.

Work management in practice has divided between development and operations toolsets. However, DevOps and Kanban are forcing a reconsideration and consolidation. Historically, here are some of the major tools and channels through which tasks and work are managed on both sides:

- Development
- Operations
- Requirements log
- Issue/risk/action item log
- Defect tracker
- Agile backlogs (user stories)
- Service desk ticketing system

Kanban is a special case, as it can mean both a manual as well as an automated system. Obviously, for time and space shifting, we need an automated system.

All of these systems have common characteristics. All can (or should) be able to:

- Register a new task
- Describe the work to be done (development or break/fix/remediate)
- Represent the current status of the work
- Track who is currently accountable for it (individual and/or team)
- Indicate the priority of the work, at least in terms of a simple categorization such as high/medium/low

More advanced systems may also be able to:

- Link one unit of work to another (either as parent/child or peer to peer)
- Track the effort spent on the work
- Prioritize and order work
- Track the referral or escalation trail of the work, if it is routed to various parties
- Link to communication channels such as conference bridges and paging systems

The first system you may find yourself acquiring along these lines is a help desk system.

RACI

RACI stands for

- Responsible
- Accountable
- Consulted
- Informed

A RACI analysis is often used when accountability must be defined for complex activities. It is used in process management, and also is seen in project management and general organizational structure.

Since it is a precursor of both process and project management, it's therefore appropriate to cover it as part of generic work management.

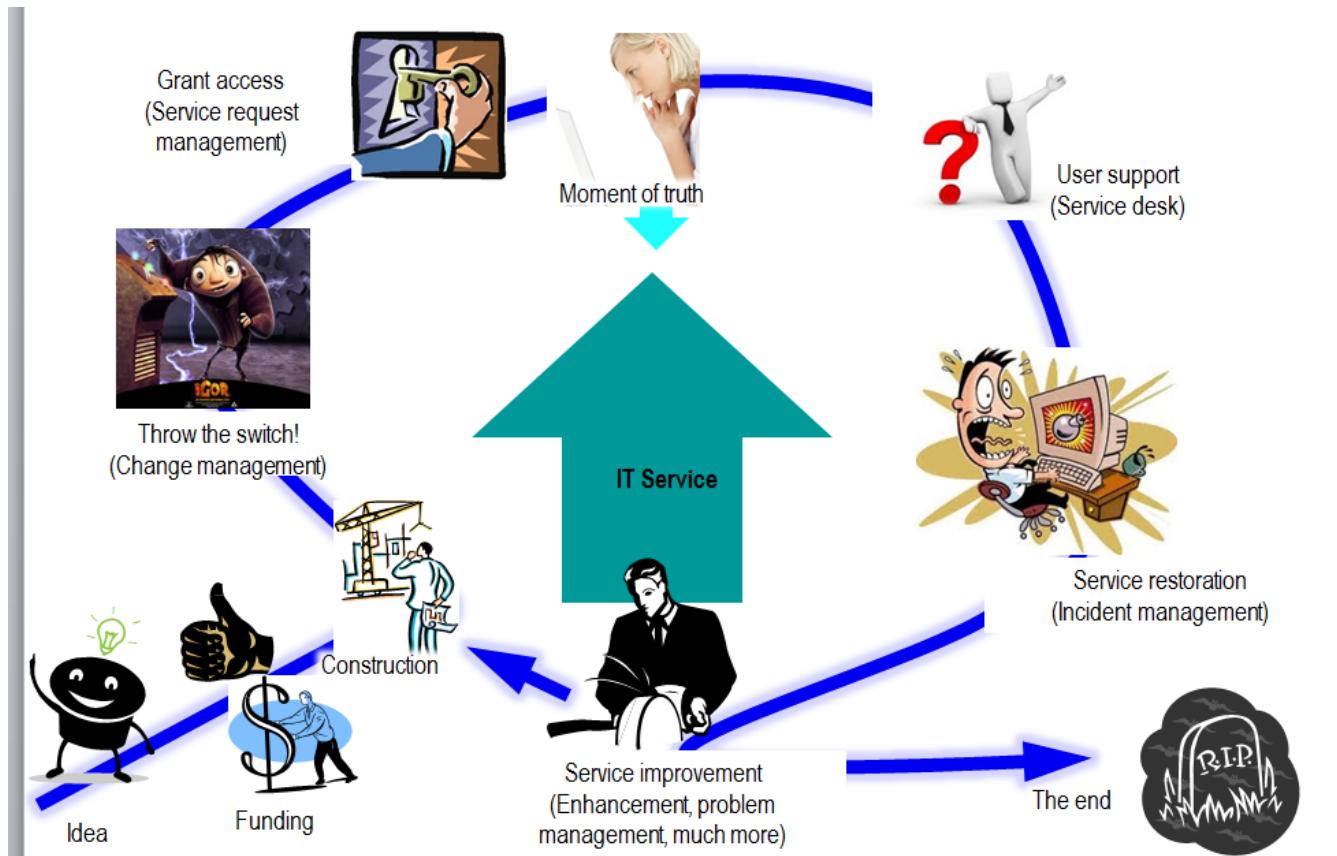
bring in reinertsen discussion

focus on specific decisions rather than abstract concepts like empowerment and strong vs. weak teams. It is much more useful for teams to know which product features they can change without outside approval than it is to find out they are "empowered."

The service desk

As a product starts to gain a user base, and as a company matures and grows, there emerges a need for human to human support. This is typically handled by a help desk or service desk.

The Service Desk (aka Help Desk), is the human face of IT when the IT systems are not meeting people's expectations. We were first briefly introduced to the concept in our Service Lifecycle:



The service desk is an interrupt-driven, task-oriented capability. It serves as a first point of contact for IT services that require some human support or intervention. As such its role can become broad, from provisioning access to assisting users in navigation & usage, to serving as an alert channel for outage reporting.

The service desk ideally answers each user's request immediately, requiring no followup. If followup is required, a "ticket" is "issued."

As a "help desk," it may be focused on end user assistance and reporting Incidents. As a "service desk," it may expand its purview to accepting provisioning or other requests of various types (and referring and tracking those requests). Note that in some frameworks, Service Request and Incident are considered to be distinct processes.

The term "ticket" dates to paper-based industrial processes, where the "help desk" might actually be a physical desk, where a user seeking services might be issued a paper ticket. Such "tickets" were also used in field services:



In IT-centric domains, tickets are virtual; they are records in databases, not paper. The user is given a ticket "ID" or "number" for tracking (e.g., so they can inquire about the request's status.) The ticket may be "routed" to someone to handle, but again in a virtual world what really happens is that the person it's routed to is directed to look at the record in the database. (In paper based processes, the ticket might well be moved physically to various parties to perform the needed work.)

- picture of service desk vs operations

In any event, a Service Desk capability needs

- channels for accepting contacts (e.g. telephone, email, chat),
- staffing appropriate to the volume and nature of those requests,
- robust workflow capabilities to track their progress, and
- routing and escalation mechanisms, since clarifying the true nature of contacts and getting them serviced by the most appropriate means are nontrivial challenges.

[more to be written e.g. provide an example of escalation]

There is extensive literature on managing Service Desks; the reader is referred first to the Help Desk Institute.

See also https://en.wikipedia.org/wiki/Help_desk.

Lean, Lean product development, and Don Reinertsen

Collaborative

Status 2016-10-08: This section needs expansion to about 3x as much content, primarily from Reinertsen and Black Swan.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Often, IT professionals resist drawing lessons from non-IT fields, because of a perception (only partially true) that these other fields (especially manufacturing) are “deterministic” while IT systems development is too “uncertain.”

In reality, manufacturing is less deterministic than IT professionals often perceive, while software development is, at its most uncertain, just another form of [new product development](#), and/or [research and development](#) and therefore can be managed on that basis.

See The Phoenix Project, Chapter 30, for an interesting discussion of these issues in an IT context [\[Kim2013\]](#).

Lean product development

We've had passing mention of Lean already in this book. But what is Lean?

Lean is the name given to a cohesive set of practices generally attributed to [Taiichi Ohno](#) and [Shigeo Shingo](#) at [Toyota](#). These practices included:

- Respect for people
- Limiting work in process
- Small batch sizes (driving towards "single piece flow")
- Just-in-time production
- Decreased cycle time

There are many, many other aspects to Lean.

IMPORTANT

Lean is important. Regardless of your intended career path, it is advisable to read the great Lean classics, including [The Machine That Changed the World](#), [Lean Thinking](#), [The Toyota Way](#), and Ohno's own [Toyota Production System](#). [Toyota Kata](#) is a more recent, in-depth analysis of Toyota's culture.

One of the challenges with applying Lean to IT (as noted previously) is that many IT professionals (especially software developers) believe that manufacturing is a "deterministic" field, whose lessons are inapplicable to developing technical products. "Creating software is like creating art, not being on an assembly line," is one line of argument.

The root cause of this debate is the distinction between product development and production. It is true that an industrial production line may be repetitive. But how did the production line come to be? It was created as part of a process of product development. **That** is not a repetitive process; it is a highly variable, creative process, and always has been.

WARNING Never confuse production with product development. The approaches, measures, and concerns are radically different. This book considers it an "original sin."

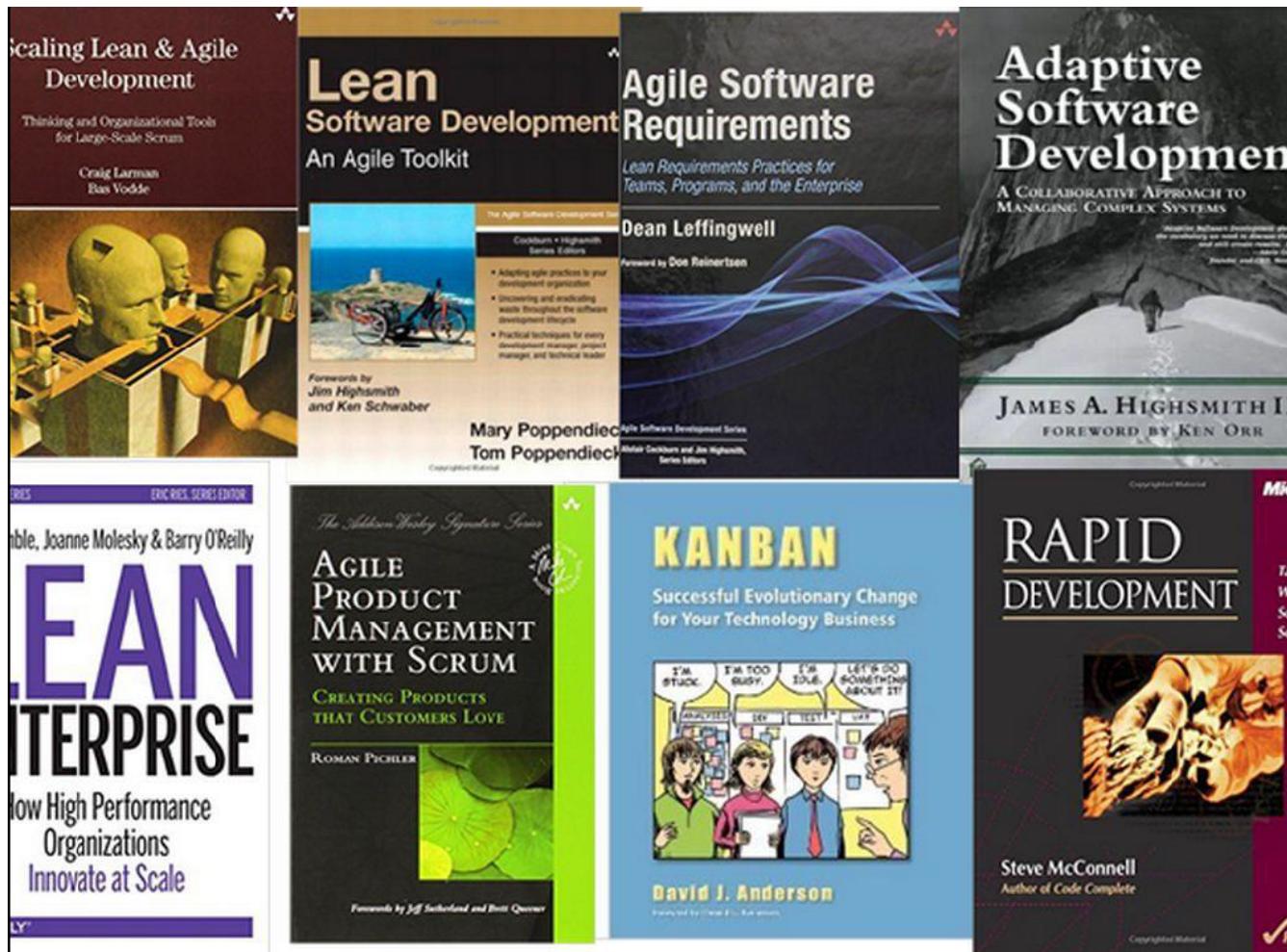
One dead end that organizations keeps pursuing is the desire to make research and development (R&D) more "predictable"; that is, to reduce variation and predictably create innovations. This never works well; game-changing innovations are usually complex responses to complex market systems dynamics, including customer psychology, current trends, and many other factors.

In IT, simply developing software for a new problem (or even new software for an old problem) is an R&D problem, not a production line problem. It is iterative, uncertain, and risky, just like other forms of product development.

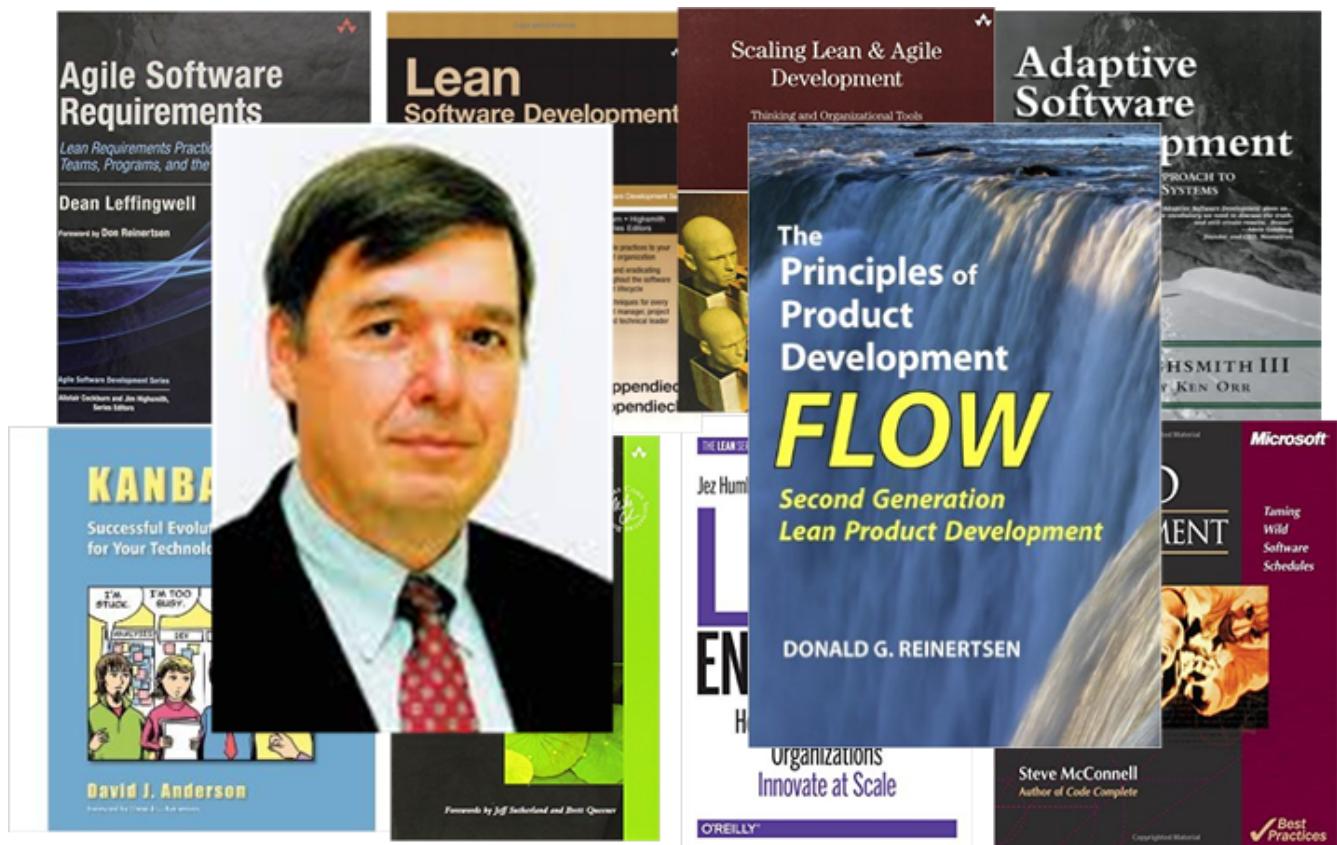
That does not mean it is completely un-manageable, or that its creation is a mysterious, artistic process. It is just a more variable process with a higher chance of failure, with a need to incorporate feedback quickly to reduce the risk of open-loop control failure.

These ideas are well known to the Agile community and its authors. However, there is one thought leader who stands out in this field: an ex-Naval officer named Donald Reinertsen.

There are many books on Agile development and management. Consider the following books, all well known and respected in the fields of software development and Agile:



What do they all have in common? They all cite Don Reinertsen:



Reinertsen's work dates back to 1991, and (originally as a co-author with Preston G. Smith) presaged

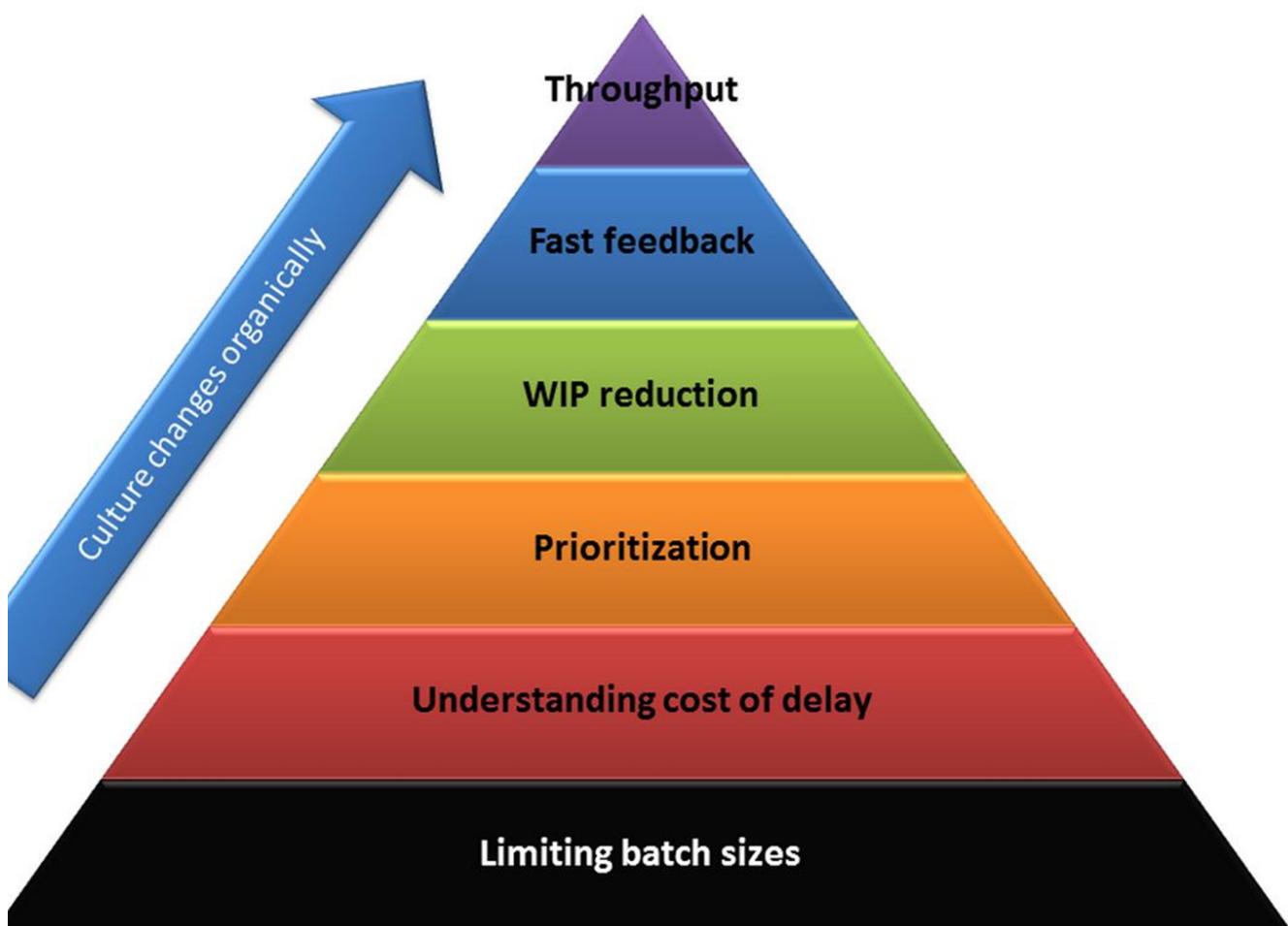
much of the Agile movement [Smith1991].

Reinertsen was partnering with David Anderson when Anderson created Kanban. He wrote the introduction to Leffingwell's *Agile Software Requirements*, the initial statement of the Scaled Agile Framework. His influence is pervasive across the Agile community.

His work is deep and based on fundamental mathematical principles such as queueing theory (lightly touched on in *The Phoenix Project*). One way of reading his work is as a series of interdependent principles:

1. The flow, or throughput of product innovation is the primary determinant of financial success.
2. Product development is essentially the creation of information.
3. The creation of information requires fast **feedback**
4. Feedback requires limiting work in process
5. Limiting work in process in product design contexts requires rigorous prioritization capabilities
6. Effective, economical prioritization requires understanding the **cost of delay** for individual product features
7. Understanding cost of delay requires smaller batch sizes, consisting of cohesive features, not large projects.

These can be summarized in the following pyramid:



[the original sin according to Reinertsen: confusing production with product development]

Cost of Delay

Don Reinertsen is well known for advocating the concept of "Cost of Delay" in understanding product economics. The term is intuitive; it represents the loss one experiences by delaying the delivery of some value. For example, if a delayed product misses a key trade show, and therefore its opportunity to launch against competitors, the cost of delay might be the entire market.

see <http://blackswanfarming.com/cost-of-delay/>

[TBW]

Beneficial variability

belongs in Ch 8 investment

OODA sidebar

After the Korean War, the US Air Force wished to clarify why its pilots had performed in a superior manner to the opposing pilots who were flying aircraft viewed as more capable.

Boyd determined that humans go through a defined process in building their mental model of complex and dynamic situations. This has been formalized in the concept of the OODA loop. Standing for:

- Observe
- Orient
- Decide
- Act

Because the US fighters were lighter, more maneuverable, and had better visibility, their pilots were able to execute the OODA loop more quickly than their opponents, leading to victory.

Boyd and others have extended this concept into various other domains including business strategy. The concept of the OODA loop is frequently mentioned in presentations on Agile methods. It is, of course, a feedback loop.

CFDs - not important at team level? Move execution concepts to section III - team of teams problems

From Plan, Build, Run to Demand, Supply, Execute

Instructor's note This is more advanced, speculative material and its presence in the book is in question. It's rough. Thoughts?

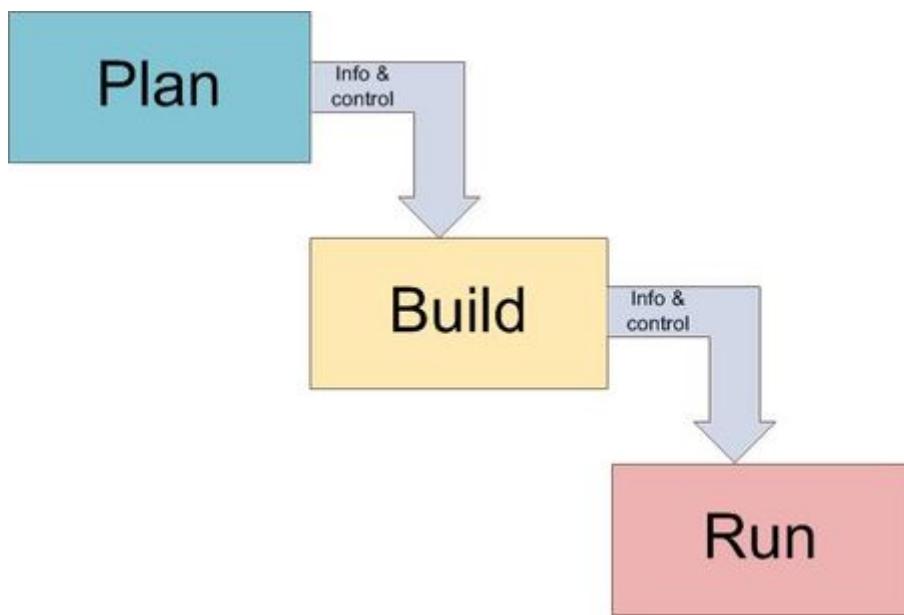
Originally published at <http://www.lean4it.com>.

Let's talk mental models. The biggest one we have in IT.

In the broad system of IT management, we often hear of three stages, "Plan/Build/Run." We make plans for a new IT system, we construct it, and we run it.

IT organizations often structure themselves and communicate to a large degree along these lines. I've used them myself in any number of writings. However, I have come to question the usefulness of "Plan-Build-Run" (which I'll abbreviate PBR) as a mental model for understanding and guiding IT management.

While it pervades IT management thinking, and can clearly be seen in the structures of frameworks like ITIL and COBIT, PBR is ill suited for the demands of 21st century IT management. I believe it is fundamentally a statement of waterfall thinking.



The major structures of Plan, Build, Run in my experience are framed as discrete, mutually exclusive states, with interfaces (information exchanges and control signalling) in between them. The model reflects IT's history of project-centric management, in which large batches of work are moved across those three phases. Since each large batch is fragile and the state transitions are risky and high ceremony, there is also concern for insulating each batch from outside "interference."

This results in systems being built in isolation from one another and without openness to feedback from later stages or other batches of work. Thus we also lose opportunities to learn from mistakes, leverage shared services and ensure architectural consistency across the IT estate.

Continuous improvement cannot be an afterthought. Yet when PBR is the fundamental model, continuous improvement activities are run as spreadsheet-driven initiatives, poorly resourced and in constant contention with the "real" work of Plan, Build, Run. (The classic example is the massive spreadsheet of Audit findings which becomes a political football in *The Phoenix Project*. Note that I advocate for a broad definition of continuous improvement.)

I'd suggest these kinds of approaches are essentially "inspecting quality in" and therefore not what Deming would have wanted. ("We cannot rely on mass inspection to improve quality.")

Finally, is PBR the reflection of your organization's communication structures? Conway's Law would suggest that PBR is the reflection of embedded organizational communications structures - dating I would argue from the time of Frederick Taylor and the worst excesses of hierarchical, command and control management philosophy.

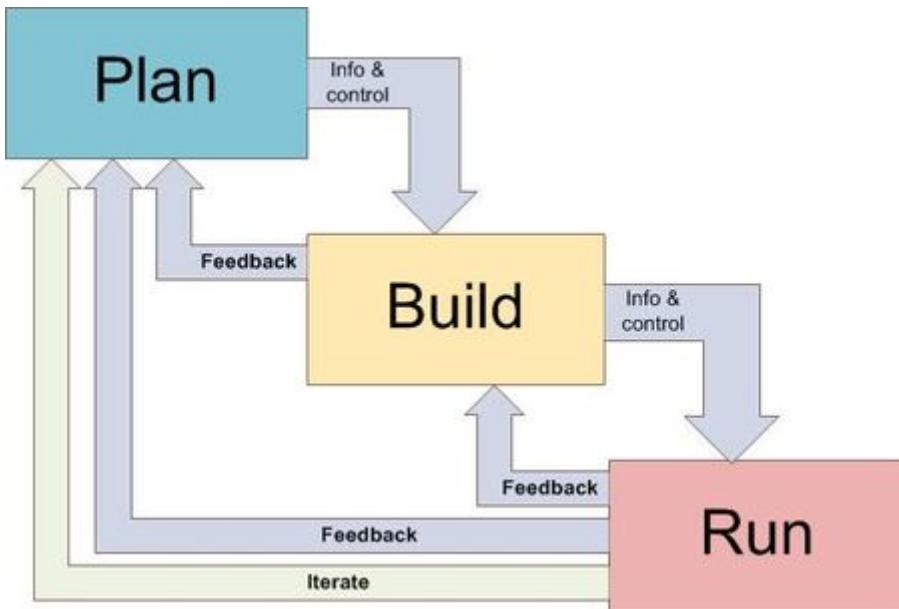
This is probably the most pernicious consequence - specialists in planning, building, and running, all trying to optimize their worlds, with the "planners" at the top, the "builders" in the middle and the "runners" taking orders at the bottom. The trouble is that this idealized hierarchical model is utterly ineffective.

It certainly doesn't reflect military principles, often caricatured as "command and control" (see Reinertsen's discussion of military doctrine on decentralized control in *Principles of Product Development Flow*). Effective product and software development has also long eschewed it; much evidence demonstrates the futility of relying on "expert" opinion and planning in truly complex domains. (See this excellent discussion from a venture capitalist at Y-Combinator.)

Concepts like Lean Startup experimentation, generalizing specialists, and empowered 2-pizza product teams do not seem to reflect a PBR model. In an Agile world, the work boundaries blur across these phases. Those who plan, also build and run, and the primary planning of worth is planning the next experiment for testing.

Some will say this is unfair. One hears various responses:

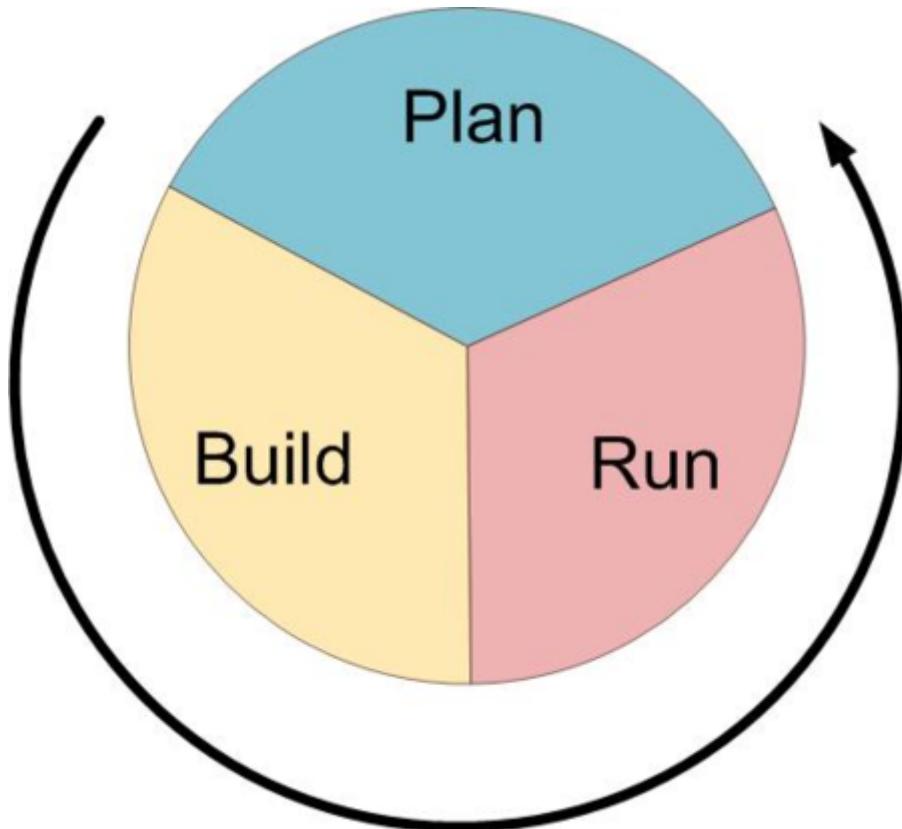
1. Iterate (with increasing speed), either the whole cycle or subsets
2. Insert feedback loops.
3. See the stages as overlapping



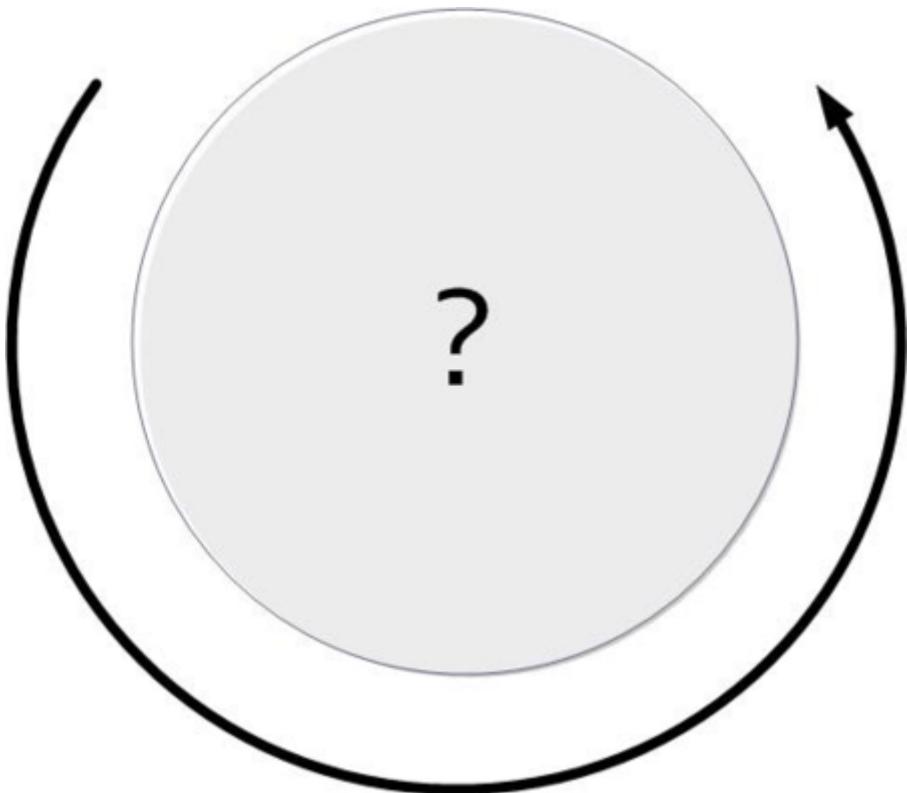
My question is, even with these enhancements, is this mental model the best foundation?

Let's look at the "fast PBR" response a bit more. Agile and Lean Startup arguably would call for many cycles of PBR, with each cycle constituting a test, or a story, or some other small grained delivery. Increasing the speed of this cycle is a widespread theme in discussions of Agile, DevOps, Continuous Delivery, and the rest.

Now if you are going to "spin it faster," a circular illustration would seem more appropriate:



But what happens when we spin this fast? Remember your elementary physics of color? It all blends together, just as we see boundaries blurring today between Dev and Ops:



And when the phases blur, what emerges? That's the question...

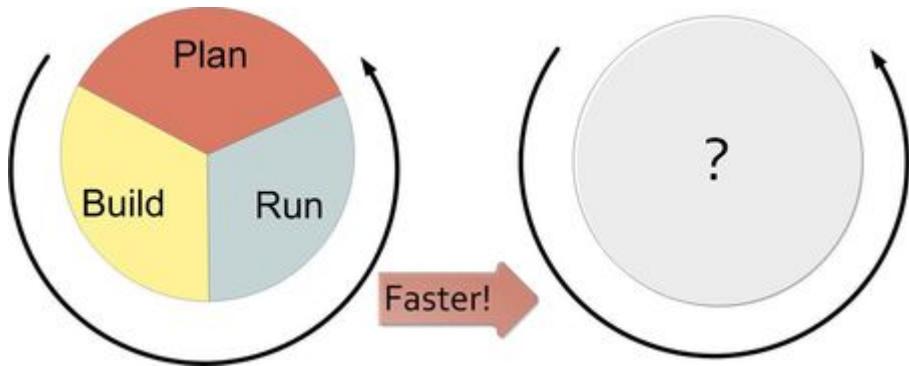
Now, I should say... IF YOU ARE LUCKY it all blends together. This assumes that you have somehow overcome the transactional overhead with those state transitions, information exchanges, and organizational handoffs, as you attempt to speed them up.

What if a given state change isn't needed for a particular cycle? You wind up fighting a system hard-wired to protect the role of each activity (and all their sub-activities). You wind up asking to "tailor" methodologies, to exempt your work from activities that are clearly non-value-add, and so forth.

Accelerating PBR isn't sufficient; it is still a sequential process with discrete steps and implications of information and control flow between each, reinforcing a dysfunctional division of organizational responsibilities. Can you get up to speed if people are trying to "optimize" the "plan," "build," or "run" phase independently? How can you optimize one third of a wheel?

I ask: is PBR how work really gets done? How value gets delivered? If it is how work gets done in your organization, how effective is it? If you are fortunate to be working in an organization using Agile approaches, can you in truth characterize them as simply "fast PBR"? Does this mental model fundamentally inform your daily experience?

I think "spin it faster" is an insufficient response. A continuous flow is needed.



When the delivery cycle accelerates and the boundaries begin to break down between Plan/Build/Run, what emerges? First, one starts to see that work has common characteristics across Plan, Build, and Run.

We might observe that a common workflow or kanban system can be utilized across planning, building, and running activities. We might also notice that smaller teams require people to cover more responsibilities ("generalizing specialists"), especially when the boundaries between planning, building, and running are blurring.

We see that distinctions between "incidents," "problems," and "user stories" become increasingly arbitrary when it is the same team that needs to respond to any. Certainly, there are differences between proactive and reactive work, but as forms of demand both compete for the same supply of resources, and when tracked in different systems cause overburden, uncoordinated effort, and bad multi-tasking.

We also see that continuous improvement activities (which are kind of a middle ground between proactive and reactive) are "just more demand" and also compete for resources with purely proactive and reactive demand.

As we consider all this, the question arises: What, in fact, **is** the IT work, in the general case? The irrevocable meeting of demand with supply, with intent to generate value. Same in IT as in the rest of industry.

These are not novel concepts. Supply and demand stem from fundamental economics and operational management. I also want to add Execution, which is a widely accepted industrial term that covers the ideally optimal translation of supply and demand into value, via detailed resource and capacity management, dispatching, process monitoring, and performance analysis.

As an aside: I am not in favor of naive approaches to equating IT delivery with manufacturing. But, as Eric says in *The Phoenix Project*,

"You think IT Operations is rocket-science compared to manufacturing. What absolute baloney... From where I'm sitting, the people in this [manufacturing facility] have been far more creative and courageous than anything I've seen come from you IT guys so far."

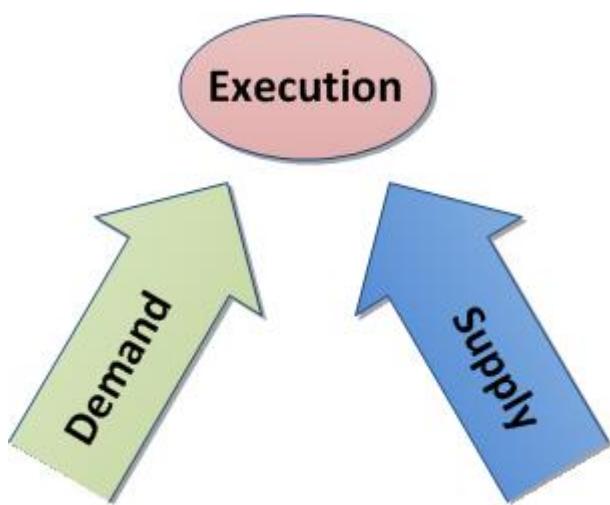
Demand management starts from the premise that regardless of the size of the implied work, all demand on IT resources should be understood in a unified manner (as popularized in the *Phoenix Project*). From a new mobile device to the day's incident reports and change requests, to a strategic initiative implying a \$10 million projects, it's all "just demand."

Different techniques come into play depending on planning horizon, value, scope, risk, and other parameters - more on this to come. Some demand signals contain others, or have complex interdependencies - e.g. an Incident that generates a Change, or a Project that contains multiple Service Requests. But if we understand demand as a unified entity we position ourselves to provide much better service to our stakeholders while at the same time giving our IT staff a saner existence.

Supply represents the fundamentals of “atoms, bits, and cells”: hardware, software, and people, under various ownership and sourcing models (e.g. Cloud). The CIO is responsible for increasingly complex IT sourcing and contract management strategies, and understanding one’s baseline supply is key to evaluating new supplier options for technology products and people with the skills to exploit them.

Finally, next generation IT execution management starts with demand and supply generally, and looks for optimal (or at least satisfactory) means of delivering value. “Projects” and “tickets” are seen as part of a unified management structure, not as the respective domains of "builders" and "runners." The availability of resources is always considered before releasing work, and ongoing scenario-based forecasting is employed to identify emergent constraints. And time tracking is completely transparent, relying on intelligent automation to determine what people have actually been working on. No Friday afternoon time reconstruction!

So, Demand - Supply - Execution. Here is a graphical representation emphasizing the continuous nature of this mental model:

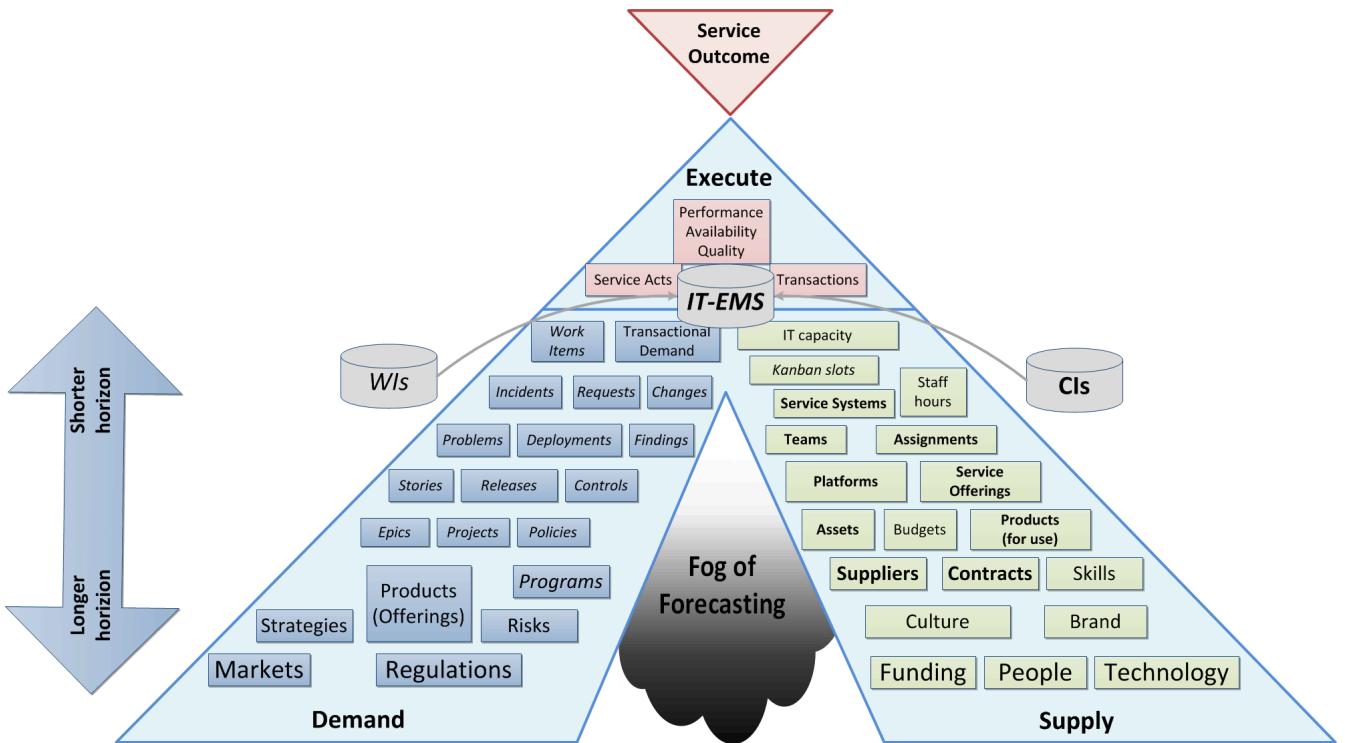


Well established IT process areas such as project, release, incident, change, and so forth are important and will continue, but I think a DSE approach could counteract the tendency to form functional silos around each—or around a particular PBR cycle and its inputs and outputs—and instead promote a whole-systems approach to IT management.

To summarize Keynes, “even the most practical man of affairs is usually in the thrall of the ideas of some long-dead economist.” Basic conceptual structures like plan/build/run and demand/supply/execute have consequences. When widely adopted to the point where they are just “common sense,” they define our social relationships, operational thinking, problem solving, and more. And thus, while we may think that “plan/build/run” is some form of IT natural law, it is a human construct that can be adapted or even discarded if we no longer find it useful. I think it’s time.

So, here is my first graphical representation of the Demand/Supply/Execute model for IT service

organizations.



The intent of this model is to provide an alternative to Plan/Build/Run and its sequential, waterfall, and Taylorist connotations. It is also intended to support modern Agile methods and philosophy which emphasize iteration, fast feedback, flow, and especially (per Don Reinertsen) managing queues, limiting Work in Progress and supporting small batch sizes.

Bottom to top, this diagram tells a story of demand and supply as they progress through increasingly refined understandings to the very specific execution of work and delivery of value.

We have markets and regulations, which define and constrain the potential demand for an IT-centric product. Markets are met with strategies and product offerings, which lead to programs of work, projects, and platform decisions. These in turn lead to identifying user stories, writing software, configuring platforms, and executing changes, service requests and work tasks.

This is NOT a methodology. These constructs can be as light or heavyweight as needed and value scenarios can originate at any point; there is no contradiction with Lean Startup and Agile principles of architectural iterations and minimum viable products. The existence of "Project" as a concept in the model does not mean that all work happens via Projects.

That finer and finer grained demand stream parallels a finer and finer grained supply stream. Large blocks of capital are translated into strategic technology choices and vendor relationships and investments in skilled people. More detailed budgets and planning culminate ultimately in the availability of people, hardware, and software for given assignments, e.g., an empty slot on a Kanban board.

Again, the journey can start anywhere, with a large block of traditionally managed programmatic capital or a small round of seed funding translated directly into a two-pizza team with maximum autonomy.

Ultimately the deployed IT service system is available for fulfilling transactional service demand

which can be measured in terms of quality, availability and performance. When demand and supply irrevocably combine, that is my definition of execution.

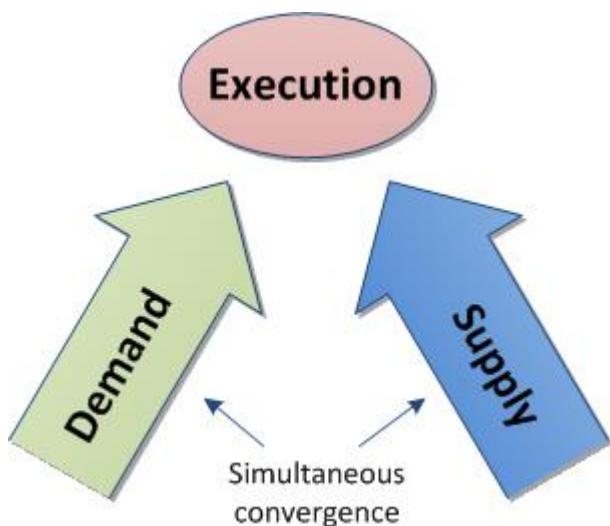
I had a number of requirements driving this model.

First, it needed to be a graphical representation that could not be read as a sequence. That is the flaw of any model which is too easily reduced to a linear format, such as Plan/Build/Run. Circular models are a popular alternative, but repeating a sequence is not enough.

My primary reservation about value networks is their lack of a goal. What is to be done? How do I develop an action plan without some sense of the network's purpose? (More here.) But the value network critique of naive sequencing is right on.

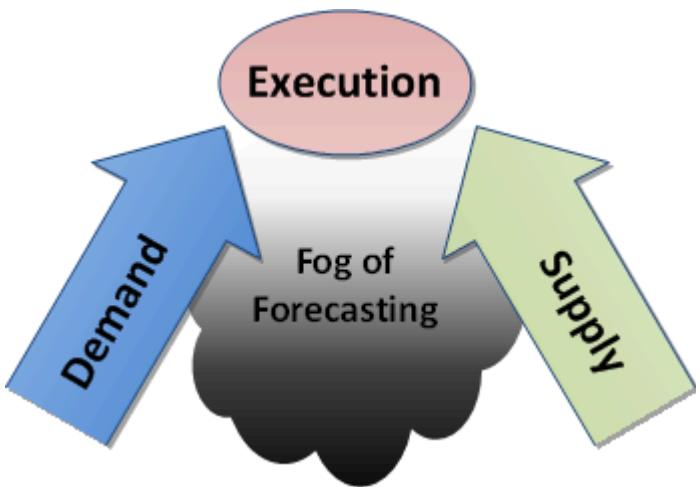
While the names "Plan/Build/Run" and "Demand/Supply/Execute" are similar in format, they are very different structurally. It is not usually possible to fully inventory demand and only THEN turn to considerations of supply. Forecasting is inherent in the relationship between the two, which both must operate continuously and simultaneously.

So, if we are trying to shift the IT mental models from sequential to network my proposal is that we start with the simplest non-sequential, non-linear concept, a vertex, two simultaneous vectors converging on a point.



Because the inverted "V" approximates human legs, it reinforces the feeling of simultaneous action. Both legs must exert effort simultaneously.

The gap between the legs of the V is filled with the "Fog of Forecasting."



With the lower level, larger grained abstractions it is more difficult to understand demand and supply, especially when product development (e.g. novel software engineering) is involved. As demand and supply converge to the point of execution, a finer and finer grained awareness is created of the impending work and whether it is likely to be successful - that is, if demand will effectively and efficiently be paired with supply. (Notice how the fog lifts as you get closer to actual execution.)

We only start to really get a feel for how execution is going to work when we get down to team and individual level assignments across all queues and ultimately actual Kanban slots or their equivalent (e.g. assigned and accepted work orders).

Turning to the choices of terms - the words inhabiting each leg (Asset, Release, Strategy, Team, etc):

Most of the nouns are things that may be found in various IT management systems (Incidents, Assets, etc), or at least deliverables & leadership conversations (Strategies, Programs). Roughly speaking, they go from larger grained and more abstract at the bottom, to smaller grained and more specific at the top.

Governance, Risk and Compliance (GRC) is included – one thing I appreciated about the Phoenix Project is that GRC is seen itself as a form of demand. Laws, regulations, and risks translate into policies, controls, and ultimately audit findings to be remediated, and all of this is also demand.

As noted above, there is still a risk that each leg will receive too sequential a reading. A value scenario can start anywhere. There is NO methodology proposed or assumed. As elsewhere, the framework simply represents frequently used IT terms, including some of my explorations of service semantics.

This picture is not overly concerned with functional boundaries (legal and/or internal). These can appear in many ways. The primary boundary is between the service act vertex (inside-out) and the service outcome vertex (outside-in). But additional org boundaries might be found between demand side and supply side functions (IT Demand Management vs. Asset Management), and/or between different abstraction levels (Project Management vs. Incident Management).

Finally, this is an IT-centric representation. The framwork might be more broadly applicable to other domains, but I will leave that to others for now.

In closing: I think this mental model is a more accurate reflection of IT practice as it is evolving. It

avoids sequential, linear and waterfall thinking as well as command and control Taylorism. It accommodates well known ITSM functional concepts, but aligned along a different fundamental structure, a structure better aligned with economic principles.

Advanced topics

===== Multi-tasking

Context switching and multi-tasking
[Human factors discussion]

Need to allocate between chapters 5, 8, 9 - or touch on & elaborate later?

Relationship between features and tasks

Decision rules
Reinertsen Boeing case

Rule engines

Synchronization

I think there is a fundamental abstract model here of tasks and synch points - OS scheduling analogs? Mutexes/semaphores? emergent dependencies?

Cadence

Cadence is an important form of synchronization. [more]

Velocity

Stories are meant to be independent and valuable. Tasks aren't. By earning velocity at the level of tasks, it is possible to be 70% dev-complete for a release without a single story being dev-complete.
[\[Narayan2015\]](#)

A look ahead to project and process

Projects: fixed time/duration/scope, not repeatable.

Processes: repeatable...

deferrable commitments

[Dan Pink, Drive: Task, time, technique, and team]

High priority: incorporate Common Ground paper
http://jeffreymbradshaw.net/publications/Common_Ground_Single.pdf

Conclusion

"Work management" may seem like an oddly general title, but it's important to distinguish the fundamental problem from more elaborate constructs such as process and project management. At the end of the day, "it's all just work."

Understanding work requires understanding tasks and their lifecycle, and the synchronization points necessary to coordinate across them.

Discussion questions

Do you choose Scrum, Kanban, both, or neither?

Research & practice

Research an automated, online Kanban tool. Compare it to a physical whiteboard. Which do you prefer and why?

Further reading

Books

- Mike Burrows, *Kanban from the Inside: Understand the Kanban Method, connect it to what you already know, introduce it with impact*
- David Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*
- Donald Reinertsen, *Principles of Product Development Flow: Second Generation Lean Product Development*

Articles So Long Scrum, Hello Kanban

- Quantifying the Cost of Delay

Chapter 6: Operations Management

Instructor's note

Although this is entitled "operations management" it also brings in infrastructure engineering at a higher level, assuming that the product is continuing to scale up. Chapter 12 will revisit infrastructure engineering and operations in terms of the most highly scaled and complex Web-scale systems.

Thus, Chapters 2, 6, and 12 constitute a sort of "infrastructure and operations" track within the book.

Introduction to Chapter 6

As your product gains more use, you find that running it inevitably becomes a distinct concern from building it. For all their logic, computers are still surprisingly unreliable. Servers running well tested software that remain "up" for weeks all of a sudden hang and have to be rebooted. Sometimes it's clear why (for example, a log filled up that no-one expected) and in other cases there just is no explanation.

Engineering and operating complex IT-based distributed systems is a significant engineering challenge. Even with infrastructure as code, it is distinct from software development per se. Questions of scalability, performance, caching, load balancing, and so forth usually become apparent first through feedback from the operations team.

Sidebar: Introducing Limoncelli.

Chapter 6 outline

- Defining "operations"
- Event management and monitoring
- Capacity and performance management
- Incidents, disasters, and everything in between
- Advanced topics
 - Ops → dev feedback
 - Configuration
 - Control
 - Web-scale IT

Chapter 6 learning objectives

- Distinguish kinds of work, especially operational vs development
- Understand basics of monitoring, event management and impact
- Describe the basics of change and incident management
- Describe operational feedback into product design & the kinds of concerns it raises
- Describe impact and dependency analysis and why it is important

Increasingly, as operational performance is defined in code (the trend to Infrastructure as Code), operational issues are equivalent to development defects.

Introducing operations management



NASA Payload Operations Integration Center

Photo credit <https://www.flickr.com/photos/nasamarshall/15330580364>, non-commercial

What do we mean by operations? Operations management is a broad topic in management theory, with whole programs dedicated to it.

We started to run into this in chapter 5, as we examined the topic of “work management” - in traditional operations management, the question of work is primary. However, in IT, “operations” tends to have a more technical meaning, being focused on the immediate questions of systems integrity, availability and performance, and feedback from the user community (i.e., the service or

help desk).

Operations often can mean, “everything but development” in an IT context. When merged with Development into “DevOps,” one gets IT Product Management with an Agile twist.

IMPORTANT In a digitally transformed enterprise, Operations is part of the Product.

Consider the following various definitions of “operations” in an IT context. Some are relevant to our “two pizza team” scenario, some might be more applicable to larger environments:

- Keeping track of an IT product’s availability (is the Web site working?), and troubleshooting if it goes down (e.g. by rebooting it).
- Answering phone calls from users requiring support. They may be calling because a system or service they need is malfunctioning. They may also be calling because they do not understand how to use the system for the value experience they have been led to expect from it.
- Physically traveling to a remote site to evaluate and if necessary update or fix IT hardware and/or software.
- Performing routine work, such as installing hardware, granting access, or running or testing backups. Such routine work may be scheduled, or it may be on request (e.g. ticketed).
- Ensuring security protocols are followed, e.g. access controls.

As above, the primary thing that operations does NOT do is develop new systems functionality. However, new functionality usually has operational impacts.

In manufacturing and other traditional industries, product development was a minority of work, while operations was where the bulk of work happened. However, when an operational task involving information becomes repetitive and risk-free, it can be automated with a computer. This continuous cycle of innovation and commoditization has driven closer and closer ties between “development” and “operations.” It also has driven confusion around exactly what is meant by “operations.”

In many organizations, there is an “Infrastructure and Operations” function. Pay close attention to the naming. A matrix may help, because we have two dimensions to consider here.

	Development phase	Operations phase
Application layer	Application developers. Handle demand, proactive and reactive, from product and operations.	Help desk. Application support and maintenance (provisioning, fixes not requiring software development).
Infrastructure layer	Engineering team. Infrastructure platform engineering and development (design and build typically of externally sourced products).	Operations center. Operational support, including monitoring system status. May monitor both infrastructure and application layers.

A day in the life

A duty-oriented, physical presence in the service of executing particular tasks is an unchanging aspect of operational responsibility. The tasks may support broader goals and strategies, and need to be systematically discharged.

What do we mean by this?

- Systems operators sitting in 24x7 operations centers, monitoring system status and responding to alerts.
- Field technicians executing work orders (install a new PC, fix a printer, service a cell tower antenna).
- Help desk representatives responding to user inquiries.
- Developers and engineers wearing pagers on a rotating basis to respond to systems outages referred to them by the operations center.

Limoncelli, Strata, and Hogan, in their excellent *Cloud Systems Administration*, emphasize the role of the "oncall" and "onduty" staff in the service of operations. Oncall staff have a primary responsibility of emergency response, and the term oncall refers to their continuous availability, even if they are not otherwise working (e.g., they are expected to pick up phone calls and pages at home and dial into emergency communications channels). Onduty staff are responsible for responding to non-critical incidents and maintaining current operations.

- Emergency vs non-emergency - it's all a matter of expectations
 - A brief look at terminology (Incident, Problem, Issue, Defect, Request)

At the end of the day, we need to remember that operational work is just one form of work. In the classic model, developers built systems and "threw them over the wall" to operations. Each side had specialized processes and technology supporting their particular concerns.

Now, companies undergoing digital transformation are experimenting with many different models; as we will see in Section III, up to and including the complete merging of Development and Operations-oriented skills under common product management.

Environments and the fierce god of "Production"

“Don’t mess with that server! It’s ... Production!!!”

— unknown

“Production” is a term that new IT recruits rapidly learn has forbidding connotations. To be “in production” means that the broader enterprise value stream is directly dependent on that asset. Breakage or mishandling will result in questions and concerns from powerful forces. Be Very Scared.

How do things get to be “in production”? What do we mean by that?

First, let’s get back to our fundamental principle that there is an IT system delivering some “moment of truth” to someone. This system can be of any scale, but fundamentally we are able to

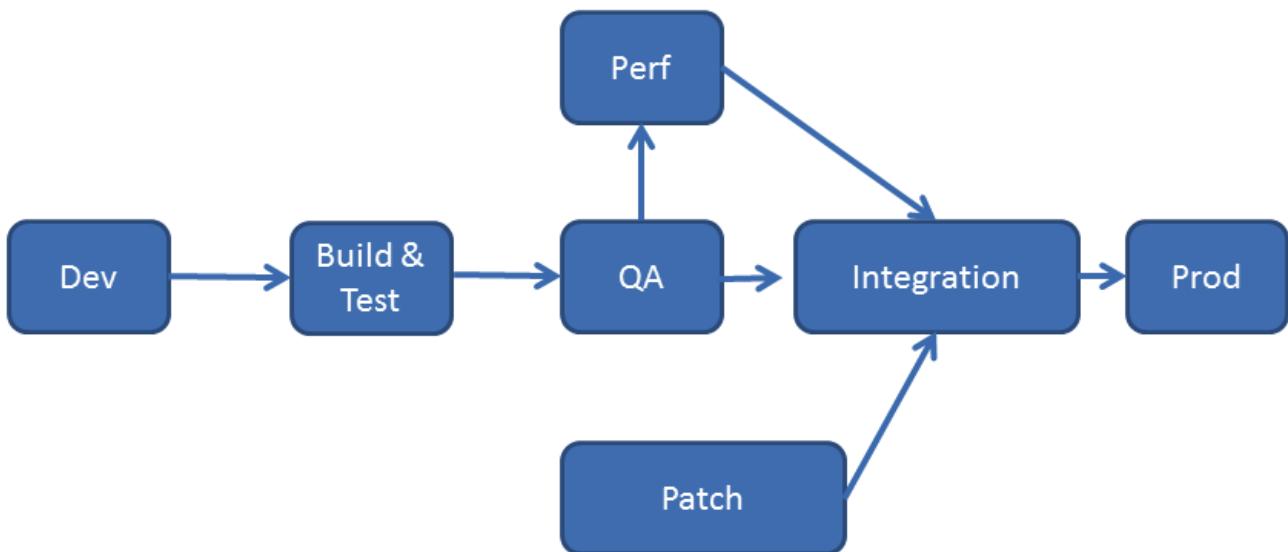
conceive of it having a “state.” (See [http://url\[State, Configuration, and Discovery\]](#).)

When we want to change the behavior of this system, we are cautious. We reproduce the system at varying levels of fidelity and experiment with potential changes. This is called development.

When we start to gain confidence in our experiments, we increase the fidelity and also start to communicate more widely that we are contemplating a change to the state of the system. We may increase the fidelity along a set of traditional names:

- Development
- Testing
- QA
- Integration
- Performance Testing

The final state, where value is realized, is “Production.”



There is nothing sacred about the environments listed above. You will see many variations, especially at scale.

The idea that one sequentially moves (“promotes”) new system functionality through a series of states to gain confidence before finally changing the state of the production system is historically well established. However, the production state is notoriously difficult to reproduce fully, especially in highly distributed environments. While infrastructure as code has simplified the problem, lower environments simply can’t match production completely in all its complexity, especially interfaced interactions with other systems or when large, expensive pools of capacity are involved.. Therefore there is always risk in changing the state of the production system. Mitigating strategies include:

- Extensive automated test harnesses that can quickly determine if system behavior has been unfavorably altered.
- Elaborating lower environments with strategies such as service virtualization to make them appear more like production.

- Hardening services against their own failure in production, or the failure of services on which they depend.
- Reducing the size (and therefore complexity and risk) of changes to production (a key DevOps strategy).
- Ensuring that changes to the production system can be easily and automatically reversed.
- Using policy-aware infrastructure management tools.

Another important development in environmental approaches is A/B testing. In this approach, the “production” environment is segregated into two or more discrete states, with different features or behaviors exposed to users in order to assess their reactions (see sidebar).

Do we need environments at all?

The author has heard it recommended that the term “environment” be eliminated, as it tends to result in turf wars and empire building, and potentially the waste of fixed assets (see chapter 8). Performance environments are particularly in question.

Instead, in a dynamic infrastructure environment (private or public), one simply defines the kind of test one wants to perform and provisions that capacity on-demand.

Testing in Production?

It used to be that the concept of “testing in production” was frowned upon. Now, with these mitigating strategies, and the recognition that complex systems cannot ever be fully reproduced, there is more tolerance for the idea.

But with older systems that may lack automated testing, incremental deployment, or easy rollback, it is strongly recommended to retain existing promotion strategies, as these are battle-tested and known to reduce risk. Often, their cycle time can be decreased.

“Development is production”

On the flip side, development systems must never be treated casually.

- The development pipeline itself (Chapter 3, figure X) represents a significant operational commitment.
- The failure of a source code repository, if not backed up, could wipe out a company (see the Code Spaces sidebar in Chapter 2).
- The failure of a build server or package repository could be almost as bad.
- In the digital economy, dozens or hundreds of developers out of work represents a severe operational and financial setback, even if the nominal “production” systems continue to function.

It’s therefore important to treat “development” platforms with the same care as production systems. This requires nuanced approaches: with infrastructure as code, particular virtual machines or containers may represent experiments, expected to fail often and be quickly rebuilt.

No need for burdensome change processes when VM base images and containers are being set up and torn down hundreds of times each day!

But the platforms supporting the instantiation and teardown of those VMs are production platforms, supporting the business of new systems development.

Monitoring

Telemetry is an automated communications process by which measurements are made and other data collected at remote or inaccessible points and transmitted to receiving equipment for monitoring. The word is derived from Greek roots: *tele* = remote, and *metron* = measure.

— Wikipedia, <https://en.wikipedia.org/wiki/Telemetry>

Computers run in large data centers, where physical access to them is tightly controlled. Therefore, we need telemetry to manage them, more typically called monitoring.

Monitoring techniques

Monitoring is the primary way we gain visibility into the systems we run. It is the process of observing information about the state of things for use in both short-term and long-term decision making.

— Limoncelli/Chalup/Hogan, *The Practice of Cloud System Administration*

But how does one “observe” computing infrastructure? Clearly, sitting in the data center (assuming you could get in) and looking at the faceplates of servers will not convey much useful information. Monitoring tools are the software that watches the software (and systems more broadly).

NOTE

Monitoring systems are similar to source control systems in that they are a critical point at which metadata diverges from the actual system under management.

A variety of techniques are used to monitor computing infrastructure. Typically these involve communication over a network with the device being managed. Often, the network traffic is over the same network carrying the primary traffic of the computers. Sometimes, however, there is a distinct “out of band” network for management traffic.

A classic, simple monitoring tool will simply observe a computing node, perhaps by “pinging” it periodically, and will raise an alert if the node does not respond within an expected time frame. If you are working through this course’s labs, you will see an example of this.

More broadly, these tools provide a variety of mechanisms for monitoring and controlling operational IT systems, often through small software “agents” deployed to computing platforms. These agents may monitor processes and their return codes, performance metrics (e.g. memory and CPU utilization), events raised through various channels, network availability, log file contents (e.g.

with standing filters for messages indicating problems), interactions with other elements in the IT infrastructure, and more.

Some monitoring covers low level system indicators not usually of direct interest to the end user. Other monitoring may attempt to synthetically re-create the user's "moment of truth." Web application response time monitoring, in which synthetic transactions are run as proxies for end user experience, is an example of this. See [Limoncelli2014], chapters 16-17.

All of this data may then be forwarded to a central console and be integrated, with the objective of supporting the organization's service level agreements in priority order. Enterprise monitoring tools are notorious for requiring agents (small, continuously-running programs) on servers; while some things can be detected without such agents, having software running on a given computer still provides the richest data. Since licensing is often agent-based, this gets expensive.

Designing software for monitoring

[to be written. Reference Nygard's Release IT; SNMP; DMTF. See also <http://blog.softwareoperability.com/.>]

Aggregation and operations centers

It is not possible for a Level-1 24 x 7 operations team to access and understand the myriads of element managers and specialized monitoring tools present in the large IT environment. Instead, these teams rely on aggregators of various kinds to provide an integrated view into the complexity. These aggregators may focus on raw status events, or specifically on performance aspects related either to the elements or to logical transactions flowing across them. They may incorporate dependencies from configuration management to provide a true "business view" into the event streams. This is directly analogous to the concept of andon board from Lean practices, or the idea of "information radiator" from Agile principles (* cite AA).

A monitoring console may present a rich set of information to an operator. Too rich, in fact, as systems become large. For this reason, monitoring tools are often linked directly to ticketing systems; on certain conditions, a ticket is created and assigned to a team or individual.

Enabling a monitoring console to auto-create tickets however, needs to be carefully considered and designed. A notorious scenario is the "ticket storm," where a monitoring system creates multiple (perhaps thousands) of tickets, all essentially in response to the same condition. Event deduplication starts to become an essential capability, which leads to distinguishing the monitoring system from the event management system.

Understanding business impact

At the intersection of event aggregation and operations centers is the need to understand business impact. It is not, for example, always obvious what a server is being used for.

This may be surprising to new students, and perhaps those with experience in smaller organizations. However, in many large "traditional" IT environments, where the operations team is distant from the development organization, it is not necessarily easy to determine what a given hardware or software resource is doing or why it is there.

Clearly, this is unacceptable in terms of security, value management, and any number of other concerns. However, from the start of distributed computing, the question “what is on that server?” has been all too frequent in large IT shops.

In mature organizations, this may be documented in a Configuration Management Database or System (CMDB/CMS). Such a system might start by simply listing the servers and their applications:

Application	Server
Quadrex	SRV0001
PL-Q	SRV0002
Quadrex	DBSRV001
TimeTrak	SRV0003
HR-Portal	SRV0003
<i>etc</i>	<i>etc</i>

(Imagine the above list, 25,000 rows long.)

This is a start, but still doesn't tell us enough. A more elaborate mapping might include business unit and contact:

Org	Contact	Application	Server
Logistics	Mary Smith	Quadrex	SRV0001
Finance	Aparna Chaudry	PL-Q	SRV0002
Logistics	Mary Smith	Quadrex	DBSRV001
Human Resources	William Jones	TimeTrak	SRV0003
Human Resources	William Jones	HR-Portal	SRV0003
<i>etc</i>	<i>etc</i>	<i>etc</i>	<i>etc</i>

The above lists are very simple examples of what can be extensive record-keeping. But the key user story is implied: if we can't ping SRV0001, we know that the Quadrex application supporting Logistics is at risk, and we should contact Mary Smith ASAP, if she hasn't already contacted us. (Sometimes, the user community calls right away; in other cases, they may not, and proactively contacting them is a positive and important step.)

State, configuration, and discovery

In computer science and automata theory, the state of a digital logic circuit or computer program is a technical term for all the stored information, at a given instant in time, to which the circuit or program has access.[1] The output of a digital circuit or computer program at any time is completely determined by its current inputs and its state.

— Wikipedia, [https://en.wikipedia.org/wiki/State_\(computer_science\)](https://en.wikipedia.org/wiki/State_(computer_science))

In all of IT (whether “infrastructure” or “applications” there is a particular concern with managing state. IT systems are remarkably fragile. One incorrect bit of information - a “0” instead of a “1” - can completely alter a system’s behavior, to the detriment of business operations depending on it.

Therefore, any development of information technology - starting with the initial definition of the computing platform - depends on robust management of state. The following are examples of state:

- The name of a particular server
- The network address of that server
- The software installed on that server, in terms of the exact version and bits that comprise it.

State also has more transient connotations:

- The current processes listed in the process table
- The memory allocated to each process
- The current users logged into the system

Finally, we saw in the previous section some server/application/business mappings. These are also a form of state.

It is therefore not possible to make blanket statements like “we need to manage state.” Computing devices go through myriads of state changes with every cycle of their internal clock. (Analog and quantum computing are out of scope for this book.)

The primary question in managing state is “what matters”? What aspects of the system need to persist, in a reliable and reproducible manner?

In terms of OODA, determining and tracking the state of a system under investigation is one of the fundamental tools. With current practices, for example, this might mean examining the configuration management/provisioning system to determine how the basic operating system was built.

Drift management

Three flavors of drift:

- Drift from standard
- Drift from baseline

- Drift from policy

forward ref to SecOps

Discovery

Closely related to monitoring is the concept of discovery. While monitoring focuses on the activity of a given system, discovery is focused on state. Answering this question has resulted in a significant market for so-called “discovery” tools, which seek to profile the server and understand its installed software, processing patterns and communications with an objective of assigning it to some management construct, such as a high level IT application or service.

What is the future of discovery tools? There is some evidence that the increasing automation of IT infrastructure will mitigate management and drift problems. As the software pipeline becomes increasingly automated, tracking software from development through production becomes easier.

Policy-based management

[review ch 2 for overlaps]

Two important concepts have emerged in the past decade: policy-based management and promise theory.

In brief, policy-based management and promise theory focus on “what,” not “how.” This may also be described as “declarative,” not “imperative” (see Sidebar).

A simple example of “declarative” vs “imperative”

Declarative: “Get me a gallon of milk.”

Imperative: “Go out the door, take a right, take a left, go into the building with a big ‘SA’ on it, go in to the last aisle, take a left, go to the third case and take the first container on the fourth shelf from the bottom. Give money to the cashier and bring the container back home.”

In configuring infrastructure, scripting is in general considered “imperative” while advanced infrastructure automation frameworks are built using a “declarative,” policy-based approach.

For example, the following shell script and Chef recipe do the same thing:
create example - can't find anything decent on web
Use directory creation example with failure of script b/c not idempotent

Capacity and performance management

Capacity and performance management are closely related, but not identical terms encountered as IT systems scale up and encounter significant load.

A capacity management system may include large quantities of data harvested from monitoring and event management systems, stored for long periods of time so that history of system utilization is understood and some degree of prediction can be ventured for upcoming utilization.



Black Friday at Target, from [Wikipedia](#)

The classic example of significant capacity utilization is the [Black Friday/Cyber Monday](#) experience of retailers. Both physical store and online ecommerce systems are placed under great strain annually around this time, with the year's profits potentially on the line.

Performance management

Performance management focuses on the responsiveness (e.g. speed) of the systems being used. Responsiveness may be related to capacity utilization, but some capacity issues don't immediately affect responsiveness. For example, a disk drive may be approaching full. When it fills, the system will immediately crash, and performance is severely affected. But until then, the system performs fine. The disk needs to be replaced on the basis of capacity reporting, not performance trending. On the other hand, some performance issues are not related to capacity. A mis-configured router might badly affect a web site's performance, but the configuration simply needs to be fixed - there is no need to handle as a capacity-related issue.

Analytics

Capacity analytics at its most advanced is a true Big Data problem domain. At a simpler level, it may consist of monitoring CPU, memory, and storage utilization across a given set of nodes, and raising alerts if certain thresholds are approached.

[examples]

As a source of demand

So, what do we do when a capacity alert is raised, either through an automated system or through the manual efforts of a capacity analyst?

There are a number of responses that may follow:

- Acquire more capacity
- Seek to use existing capacity more efficiently
- Throttle demand somehow

Other operations techniques and tools

[to be written]

- Communications channels
 - Conference bridges
 - Instant messaging & chat
 - Email
 - Cell phone
 - Paging
- Record keeping
 - Runbooks/playbooks
 - Running incident logs
- Drills
 - Simian army
- Post-mortems

Advanced topics in operations

A critique of the "NoOps" idea

Human in the loop problem

The feedback loop from ops to dev

Re-visiting dual-axis

Classes of work

A deeper look at measurement

Deeper theory

- * States and sense-making and risk management around state transitions
(Systems that are too big, complex, and fluid to be considered as having a "state"???)
- * Complexity & uncertainty
- * Promise theory?
- * Burgess measuring system normality paper

<http://perfcap.blogspot.com/2012/03/ops-devops-and-noops-at-netflix.html>
<https://gist.github.com/jallspaw/2140086>

<https://newtraell.cs.uchicago.edu/research/publications/techreports/TR-2016-03>
<http://www.kitchensoap.com/2013/09/30/learning-from-failure-at-etsy/>
<http://www.kitchensoap.com/2013/10/29/counterfactuals-knight-capital/>

Conclusion

While operations management is an integral part of "DevOps," and of a given product's management, it is still a distinct area of concern and practice. The focus on interrupt-driven work results in different personality types being attracted to it, and different management approaches required.

Operations management has a broader industry meaning, but in the narrower IT sense, includes concerns of monitoring, state and configuration management, capacity and performance concerns, and related topics.

Discussion questions

- Have you or anyone you've known worked in a call center or operations center? What was it like?
- Do you prefer interrupt-driven or focused development work?

Research & practice

- Stand up two VMs, one with a web server, the other with Nagios or another open-source monitoring tool. Experiment with monitoring various aspects of the web server.
- In a team kanban setting, experiment with both user stories and operations requests coming into the same board or product team. What issues arise? How can you best handle the diversity of work? What if there were 10 times as much, what would you do?

Further reading

- Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan, [The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Volume 2](#)
- John Allspaw, [Web Operations: Keeping the Data On Time](#)
- Sidney Dekker, [Field Guide to Understanding Human Error](#)
- Sidney Dekker, [Drift into Failure](#)

Articles/posts

[AWS DynamoDB Fall 2015 outage](#)

[The truth behind building multiple environments to run your services](#)

[Orienting in Virtual Perceptual Fields](#)

http://www.allofe.com/gen/corp_generated_bin/documents/basic_module/designing1.pdf

[The Netflix Simian Army](#)

[Beyond DevOps - How Netflix Bridges the Gap](#)

[Continuous Integration Platform using Docker Container: Jenkins, SonarQube, Nexus, GitLab](#)

[Hiring a Tech Ops Team work into chapter](#)

Section II Conclusion

In this section, we considered the basic elements necessary for a collaborative product team, achieving success but still at a manageable human scale.

Our IT-centric team needed capabilities for evolving its product, managing its work, and operating its product. In some cases, time and space shifting might drive the team to automating basic capabilities such as work management and ticketing. The overall assumption however was that for the most part, people are co-located and still can communicate with minimal friction.

Section III: Team of Teams

Notes on second half of book... as of 2016, this is where the conflict between Agile/Lean IT and the old ways becomes more pronounced ... the first half of the book represents the new industry consensus, but as we seek to scale it up, everywhere one looks one sees unquestioned acceptance of Taylorist approaches unsuited to Lean product management in a services organization ...

Scenario

You are now a "team of teams," at a size where face to face communication is increasingly supplemented by other forms of communication and coordination. You are seeing more and more specialization in your organization, with the tendency of functional specialists to identify more with their field than with the needs of your customers and your business.

Your teams are all good, and get results, but in different ways. You need some level of coordination and not everyone can talk to everyone; people are no longer co-located and there may be different schedules involved.

You now have multiple products. As you moved from your first product to adding more, significant organizational evolution was required. Your products may also be developing complex interdependencies. Tensions between various teams are starting to emerge. There is an increasing desire among your stakeholders and executives for control and predictability. Resources are limited and always in contention. You are considering various frameworks for managing your organization. This is where it gets hard.

New employees are bringing in their experiences, and the more experienced ones seem to assume that the company will use "projects" and "processes" to get work done. There are no shortage of contractors and consultants all advocating various flavors of process and project management, some advocating older approaches and "frameworks" and others proposing newer Agile & Lean perspectives. However, the very concepts of process and project management are occasionally called into question by both your employees and various "thought leaders," and it's all very confusing.

We need to understand where these ideas came from, how they relate to each other, and how they are evolving in a digitally transforming world.

Welcome to the coordination problem.

etymology of co-ordination?

Section III introduction

Scenario:

Chapter 7: Culture and Organization We're getting big. How are we formally structured? How are

people grouped, and to whom do they report, with what kind of expectations? Why this way and not that? Who are we? What are the unspoken assumptions that underly our daily work?

Chapter 8: Project and Resource Management You want to get bigger stuff done. Some argue that you don't need projects at all any more, that it's all about **products**, not **projects**. But you've seen that your most ambitious ideas require orchestration. And when resources are tight (and when aren't they?) their consumption must be governed somehow. Financial management has always been a concern, but it needs increasing formalization as your activities diversify.

Furthermore, do we have the right people doing the the work? How can we find the best people to grow our initiative? We also remain concerned that work continues to flow well, that we don't take on too much work in process, and that people are not overloaded and multi-tasking.

Chapter 9: Process Management OK, we have a structure. Work needs to flow across the structure, and the projects and products need certain resources and services delivered predictably. Our increasingly specialized organization needs some specialization of processes, and without some kind of coordination our risks increase along with our complexity.

Process management, like project management, is a discipline unto itself and one of the most powerful tools in your toolbox. You start to realize there is a process by which process itself is managed - the process of continuous improvement.

Special section: The elaborated IT organization

The laundry lists supporting functions (sourcing...)

IVI IT4IT

The second half of this book will devote increasing attention to frameworks. What is a framework? A framework is a formalized body of knowledge, usually sustained by some appropriate professional body, that defines the necessary and appropriate skills and concepts for practitioners.

The perils of "best practices"

The broken ITOM

The IT senior executive

vision, framework, governance mechanisms... a day in the life

Creating the organizational mission and goals that provide the framework for selecting projects
Setting policies and procedures for addressing priorities and conflicts
Creating and maintaining labor and financial information systems
Providing facilities and equipment to support project work
Defining the limits of managers' decision-making authority
Helping to resolve project issues and decisions that can't be handled successfully at lower levels in the organization

Portny, Stanley E. (2013-04-09). Project Management For Dummies (Kindle Locations 5466-5467). Wiley. Kindle Edition.

Chapter 7: Organization and Culture

Introduction to Chapter 7

Roughly speaking those who work in connection with the [Automated Computing Engine] will be divided into its masters and its servants. Its masters will plan out instruction tables for it, thinking up deeper and deeper ways of using it. Its servants will feed it with cards as it calls for them. They will put right any parts that go wrong. They will assemble data that it requires. In fact the servants will take the place of limbs...

As time goes on the calculator itself will take over the functions both of masters and of servants. The servants will be replaced by mechanical and electrical limbs and sense organs... The masters are liable to get replaced because as soon as any technique becomes at all stereotyped it becomes possible to devise a set of instruction tables which will enable the electronic computer to do it for itself..."

— Alan Turing, Lecture on the Automated Computing Engine -- 1947

You are going through a critical juncture, the "team of teams" transition. You have increasingly specialized people delivering an increasingly complex product, or perhaps even several distinct products. You are in constant discussions around the tension between functional depth versus cross functional delivery.

Many of your employees and consultants emphasize the role of culture, but what do they mean? Is there such a thing as a "good" culture? How is one culture better than another? How can you retain that startup feel you had previously, when things are getting this big?

You often think about how your company should be structured. There is no shortage of opinions there either. From functional centers of excellence to cross-functional product teams, and from strictly hierarchical models to radical models like holacracy, there seems to be an infinite variety of choices.

Creating conditions...

Chapter 7 outline

- Why organization matters

- A simple organizational transformation
- A look at some of Mintzberg's ideas on organization
- A deeper look at product versus function
- Organizational culture
 - Rother's research on Toyota Kata

Chapter 7 learning objectives

- Distinguish between functional versus product organizations
- Gain an introductory understanding of Mintzberg's "five forces"
- Understand various concepts of culture

Why organization matters

In the early stages, you have to hire generalists who are both willing and able to take on dozens of tasks at once. Your developers will have to speak with potential customers; your accountants will have to give advice on product direction; and the born salesman on your team will need to put the phone down a few hours a day and set up a new employee's computer. This is the exciting, four-people-and-an-idea stage popularly associated with startups—but it doesn't last very long.

For a lot of your employees, growing out of this phase will be a welcome development: programmers don't want to be in accounting meetings, and salespeople don't want to sit in a dark, quiet room with the engineers. People have talents and skills they want to develop, and a healthy degree of specialization allows them to do that.

— Matt Blumberg, *Scaling Up*

So, you are getting bigger now.

Even as a cohesive, single team you had increasing complexity of support services. You needed legal and accounting advice to get the startup going. When you started hiring people, you needed HR and payroll, and at some point you need an internal person whose daily job is money (they'll become the CFO someday).

You also are buying stuff and paying bills, and you've got sales people and marketing people and you need to support your customers and collect money they have promised to pay.

How are you going to organize? More importantly, how are you going to think about organizing? This is a hard question. It's important to get organization right, and the question never goes away.

There is a lot of theory, history, and advice available on to how to organize your company.

This is not a general business school textbook, or even a textbook on startups and entrepreneurship. However, in the current environment, it is essential for technologists, no matter how specialized, to understand the general context they are operating in.

In one chapter, we cannot cover much, but there's an increasing consensus that the choice of organizational form has a fundamental impact on the success of IT-based product delivery.

In keeping with our [evolutionary approach](#), let's assume you've been fairly ad-hoc in your organizational structure up to now, doing your best to avoid specialization. Perhaps you've even been working as a collective. Nevertheless, you've needed a variety of skills to get this far in your journey: you are certainly not all Java programmers!

Perhaps the first question you need to ask is to what extent you wish to centralize IT. This is a significant one. The current trend is towards decentralizing IT, but there are good reasons to maintain IT as a separate organizational entity. Sriram Narayan, in his book *Agile Organization Design* ([\[Narayam2015\]](#)), points out that "IT work is labor-intensive and highly specialized," and therefore sourcing IT talent is a particular organizational capability it may not make sense to distribute. Furthermore, he observes that IT work is performed on medium to long time scales, and "IT culture" differs from "business culture." (The last may be debatable in the long run, in the digitally transforming world.)

Centralized versus decentralized IT

What do we mean by "centralized" versus "decentralized," anyways? There are many models for structuring IT, but at one end of the spectrum is when everyone doing anything significant with configuring computers, developing software, or supporting IT networking reports up through a powerful Chief Information Officer, who may report to the CEO, COO, CFO, or other senior executive.

By contrast, in a "decentralized" model, computing is procured and managed by diverse managers in other functional units. So, the Human Resource VP might have an IT manager, the Marketing director would have one, and so on. The IT managers might know each other but they report to different managers and may have a very limited ability to share resources and learnings.

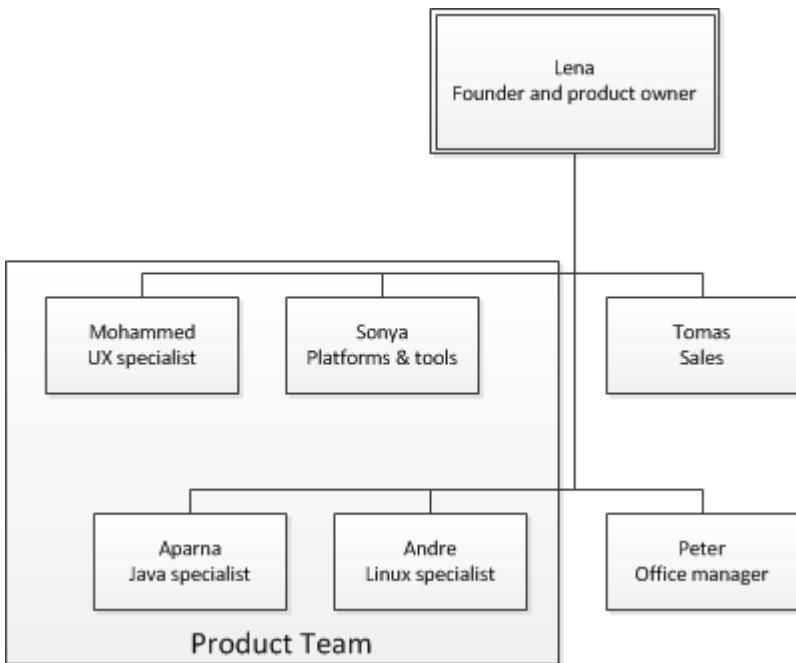
As noted at the [outset](#), IT has certain unchanging aspects regardless of whether is a separate or embedded organization. However, the purpose of this chapter is to help you think carefully about this very question.

Organizational transition and the Spotify model

Everyone in a startup wears many different hats. The first step in scaling your business is to list all the different hats you and your team members are wearing.

— Furr and Ahlstrom, Nail It then Scale It

Perhaps your organizational form at the end of Section 2 looks like this:



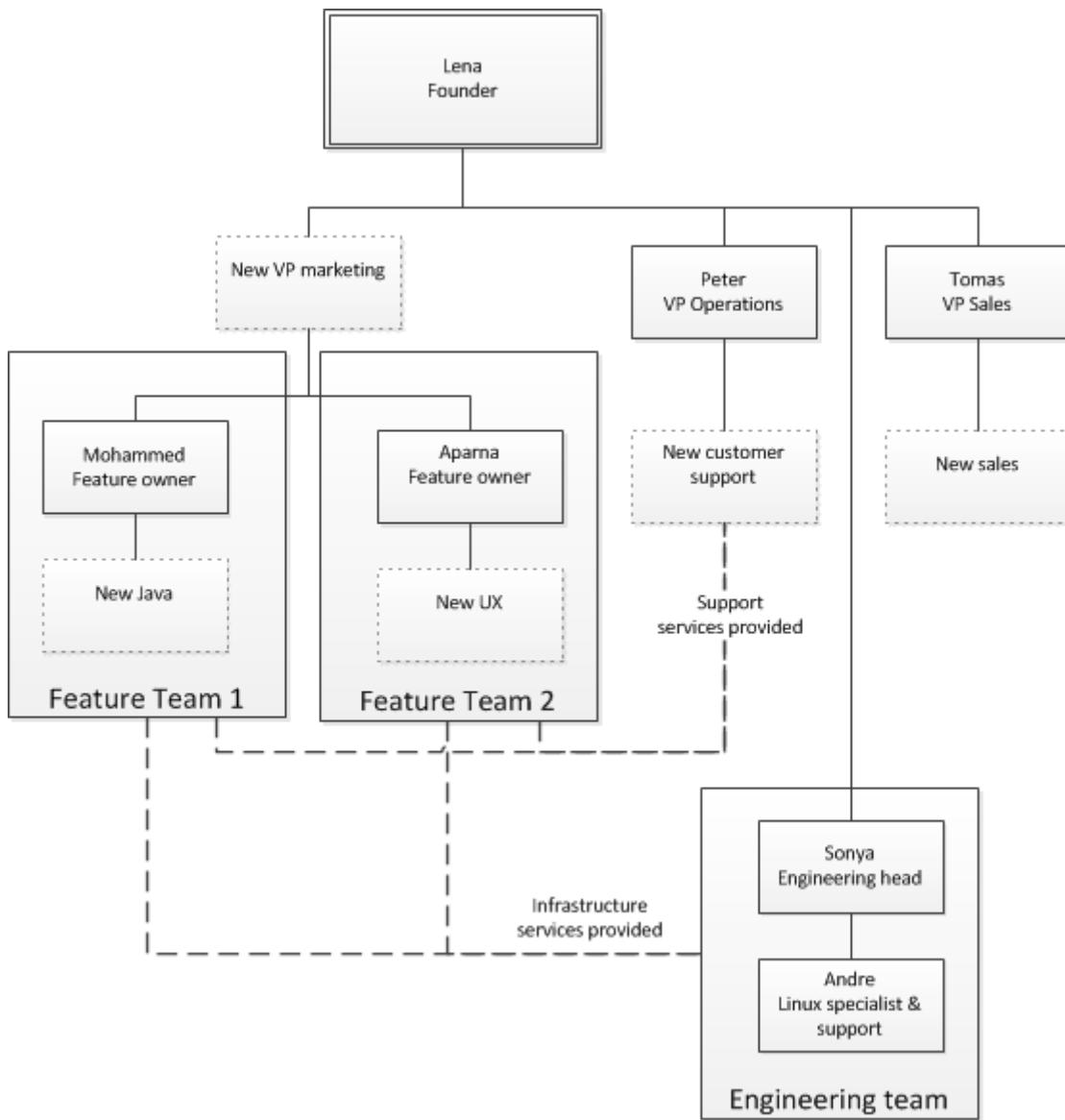
You have the following personnel:

- Lena, the visionary and product owner, who had the idea for the startup. Lena retains the overall vision for the product and is therefore the Chief Marketing Officer as well as CEO.
- Mohammed, the user experience specialist
- Aparna, the Java software engineer
- Sonya, who has a broad scope of platforms and tools, including the development pipeline, source control, and the product's NoSQL database
- Tomas, the sales lead.
- Peter, the office manager, who is responsible for a range of operational, financial, and lightweight HR duties.

Because of your success, Lena needs to go from one team to two. She is cautious about this transition, has heard that there are a few different ways in which this can be done, and the decisions may have long term consequences!

Let's assume you are going to break the product into two feature teams. One will focus on the existing market and the other will develop a new major feature for a new, distinct set of user value propositions.

In order to do this, you are going to bring in a new UX specialist, a new Java programmer, a new sales person, and someone to field customer support calls. Lena also intends to step back a bit from direct product ownership, so a new Chief Marketing Officer comes in. She talks with an organizational consultant who proposed the following new structure:



Proposed structure #1

What's the difference between Sales and Marketing? Marketing tells a story that spreads. Sales overcomes the natural resistance to say yes.

— Seth Godin, http://www.ehow.com/info_8088140_cons-combining-marketing-sales.html

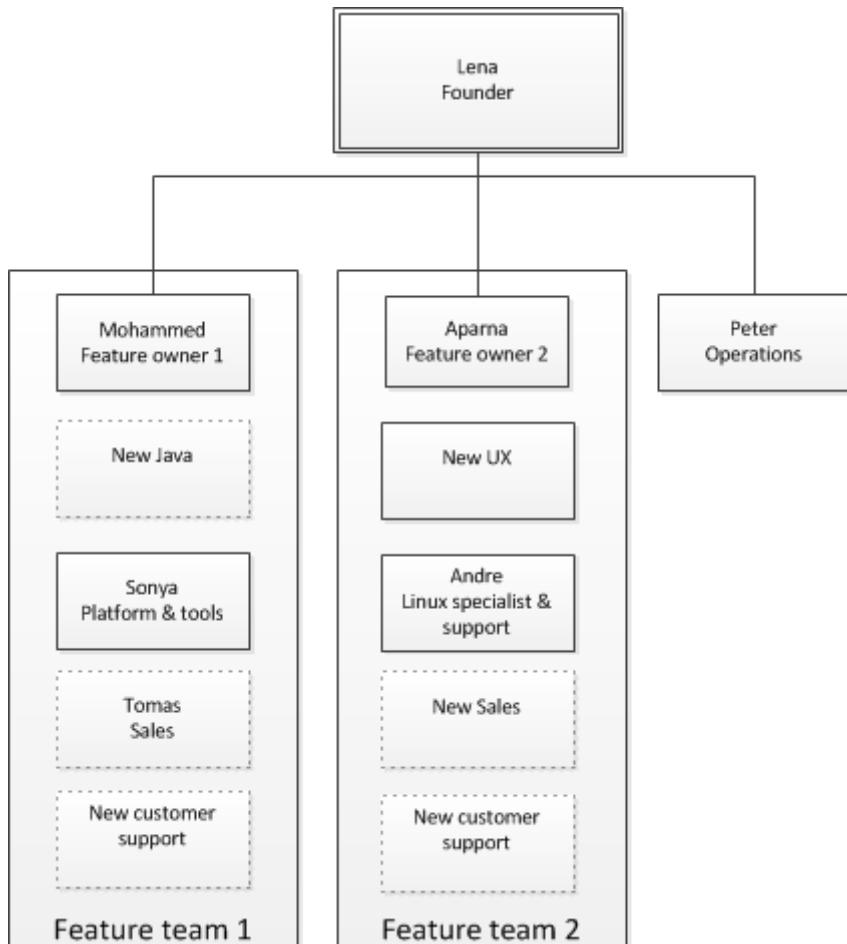
This may seem reasonable. Everyone agreed that there is no need to add more people to help Sonya and Andre; they can support both feature teams. Obviously, the new sales person comes in under Tomas. Because Peter was the one receiving all the calls, and in the interest of giving him a clear career path, it was determined that the first product support person would come in under him, in what was anticipated to become a dedicated help desk/call center.

It was perceived that the engineering team would likely be receiving various requests and so a dotted line appears on the organization chart, indicating a service relationship. The new support person under Peter would also be shared across both teams, and there needed to be some means of communicating support requests to the feature teams when their support was needed. There was some question as to how the office Kanban boards might be reconfigured to support these relationships. Peter and Sonya, after some online research, started to become interested in something called ITIL (the IT Infrastructure Library), which prescribed multiple IT processes that

could be used as a basis for organizing.

Proposed structure #2

But before that discussion continued, Lena retained another consultant as a cross-check. This consultant had worked with many Silicon Valley startups and had a very different opinion. This consultant was inspired by the “[Spotify model](#)” and proposed the following scaled-down version of it:



Sidebar: Dunbar and Spotify Managing organizational growth is a well studied problem. Military examples are often seen, as military organizations have devoted much resource to understanding organizational dynamics.

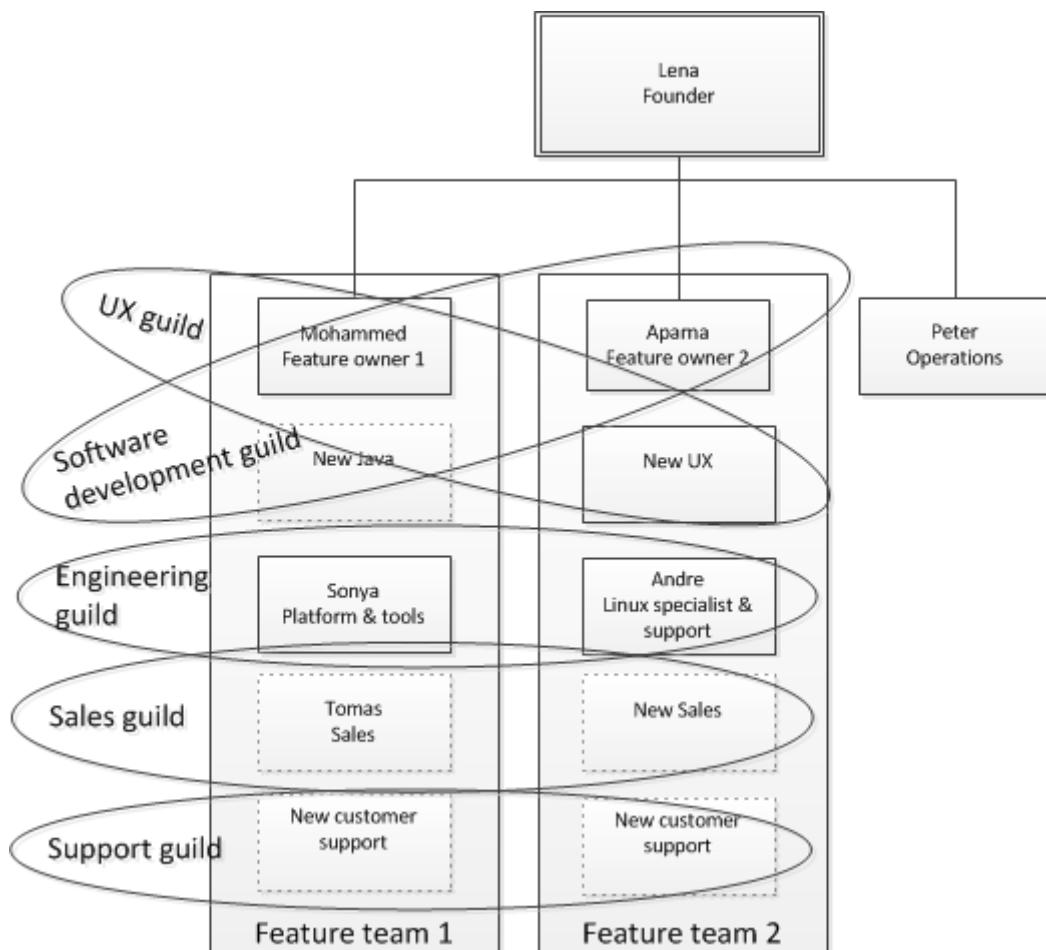
There is much research into how human beings organize into groups of various sizes. Perhaps the best known is that of Robin Dunbar, who has theorized that human beings (as evolved primates) can “only comfortably maintain 150 stable relationships.” (per [Wikipedia](#))

Dunbar has researched this projection across the anthropological data, finding that human groups tend to fall into size ranges of

- 30-50 (bands)
- 100-200 (clans)
- 500-2500 (tribes)

Online music vendor Spotify has adopted insights from Dunbar as it scales.

The consultant also added something called “guilds” to the model:



The “guilds” are not formal reporting mechanisms, but rather a way for people doing similar work to share experiences, best practices, and so forth. Each tribe has a designated lead:

- Tomas leads the Sales guild

- Mohammed leads the UX guild
- Aparna leads the Software Development guild
- Sonya leads the Engineering guild
- TBD leads the support guild

The idea would be that the guilds would meet regularly (perhaps bi-weekly) and would have their own forums on the company's social media channels. The head of each guild would meet with Lena monthly to discuss issues and concerns, and would have a direct channel of communication to her as needed for more time sensitive matters.

not sure there is any evidence for a Sales "guild" even in smaller firms - thinking it tends to be a contained function everywhere. Seeking input.

NOTE

The strict Spotify model calls for "tribes" that work on products and features, with "chapters" within each tribe for the common interests. The chapters in turn meet as "guilds" across the "tribes." The model presented here is scaled down and omits the chapter concept in the interest of simplicity, as there would be "chapters" of one.

These two organizational structures were very different and were extensively discussed in the company (Lena preferring a transparent, open decision making process.) The following arguments pro and con were noted:

	Pro	Con
Option 1	<p>Groups specialists together, so they can share knowledge and become excellent at what they do.</p> <p>Perceived to be more scalable, with the infrastructure and support teams loosely coupled to the product teams, and the establishment of the company's first true support and shared services processes.</p>	<p>Concern that the new process approach would lead to delays and finger-pointing.</p> <p>Concern that the engineering specialist team in particular would start to lose focus on customer needs.</p> <p>More expensive, with a new VP of marketing.</p>

	Pro	Con
Option 2	<p>Groups product teams together and assigns each their own engineering and support resources, so they can most effectively understand the customer needs as a team and organize themselves to meet those needs.</p> <p>Less expensive, substituting a new customer support person instead of a VP.</p>	<p>Specialists are not working together and exchanging key knowledge and experience daily (the guilds meet at most weekly).</p> <p>Concern that engineering standards might become fragmented and weakened if the product teams went in different directions.</p> <p>Concern that Feature Team 2 did not need a support person immediately.</p> <p>Concern that Peter no longer had a clear career path as VP of Operations.</p>

One fact everyone agreed on was that Sonya had to do more Linux and Andre has to learn some NoSQL and other platform technology. But this fact was not put into the matrix as there was not agreement as to whether this was a pro or a con. On one hand, it did seem a bit wasteful, but it was also recognized as a good thing in terms of cross training and deepening the bench. Sonya also felt that a Data guild would eventually be needed.

Everyone agreed that the engineering guild in particular would need to be a very strong guild, to ensure consistency of approaches around key disciplines like source control, security, platform choices, build pipeline, and so forth. The other guilds could be a bit lighter weight, but the engineering guild leader could set binding technical policies across the product teams if need be (the expectation was that this would not be done lightly).

It was recognized that eventually a distinct operations & engineering team might still be necessary, but probably not until another round of scaling, and that that team, per the Spotify model, would be more focused on setting up self-service tools for the product teams, and would avoid ticketed work as much as possible.

Another “neither pro nor con” but important was that the second option eliminated the new VP layer. Establishing an executive layer might still happen later, the consultant suggested, but it was premature to do so now.

After further discussion and benchmarking with other startups, the second option was chosen. The new support person for Feature Team 2 would be able to be hired without urgency, allowing time for a high quality search.

This hypothetical case illustrates many of the themes we will explore throughout the rest of this chapter:

- Traditional functional vs product-centric model

- Influence of Spotify approach

Organizational theory

Five coordinating mechanisms seem to explain the fundamental ways in which organizations coordinate their work: mutual adjustment, direct supervision, standardization of work processes, standardization of work outputs, and standardization of worker skills.

These should be considered the most basic elements of structure, the glue that holds organizations together...As organizational work becomes more complicated, the favored means of coordination seems to shift from mutual adjustment to direct supervision to standardization, preferably of work processes, otherwise of outputs, or else of skills, **finally reverting back to mutual adjustment.** (emphasis added)

— Henry Mintzberg, Structure in Fives

As mentioned at the outset of this chapter, there is much research and thought available on the question of how to organize an enterprise.

One useful model for understanding the problems of organizational structure is the Five Forces model of Henry Mintzberg [[Mintzberg1983](#)]. While much organization theory (even recent work) is presented as a simple, sequential model, the Mintzberg approach is more sophisticated.

In simple, sequential models, organizations are presented as evolving from less good to more good types. Such models might propose a simplistic, bad-better-best sequence such as:

Type 1 Simple, authoritarian and fear-based

Type 2 Strictly hierarchical and complex

Type 3 Divisionally decentralized

Type 4 Matrixed and process-driven

Type 5 Evolutionary/next-generation

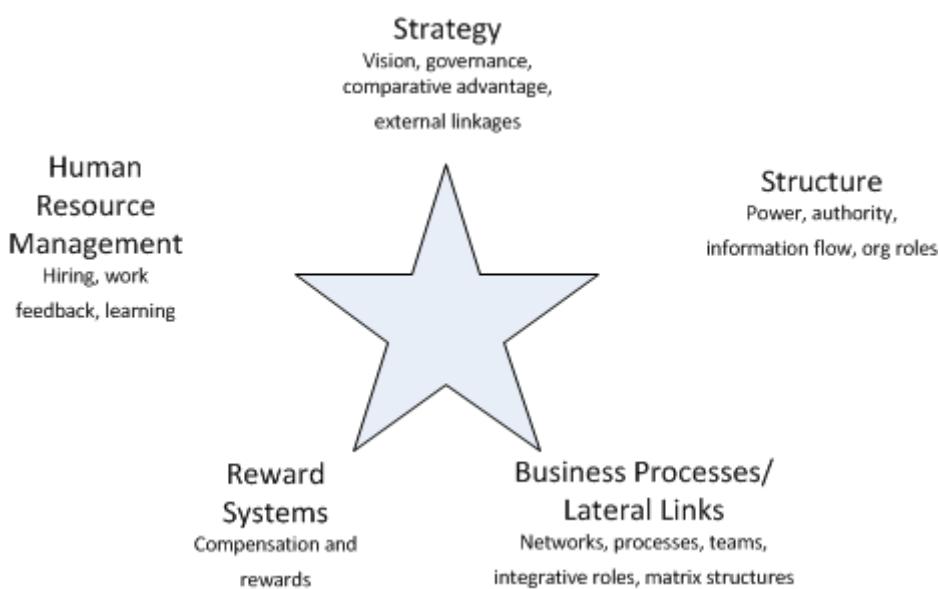
(The above sequence is a synthesis of various industry sources which are deliberately left unidentified, to protect the guilty.)

However, Mintzberg points out that different industry sectors and business types result in different kinds of organizational forms, and that certain tendencies (“pulls”) are evident which are not in and of themselves either good or bad.

Factors like line/staff interactions and lateral linkages also complicate the picture. After extensive review and summarization of peer-reviewed research, Mintzberg developed the following models of corporate structure.

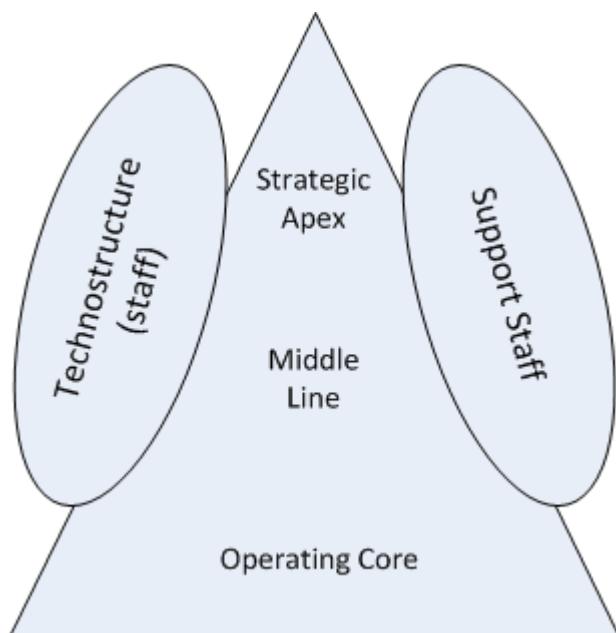
First, the question of structure is at an equivalent level with various other internal concerns:

- Strategy
- Structure
- Human Resource Management
- Reward Systems
- Business Processes & Lateral Links



Adapted from [\[Mintzberg1983\]](#)

Within the particular domain of structure, Mintzberg identified five major parts existing in most organizations:



Adapted from [\[Mintzberg1983\]](#)

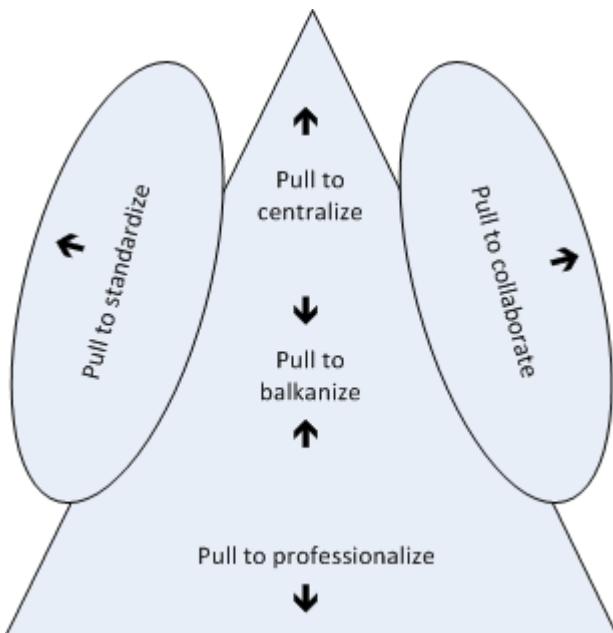
The five major areas are:

- Strategic Apex
- Middle Line
- Operating Core
- Technostructure
- Support Staff

Definitions and examples for business and IT:

Area	Definition	Mainstream example	IT example
Strategic Apex	The owners and their executive representatives	Board of directors, CEO	CIO & direct reports
Middle Line	Those who manage the Operating Core	Sales executive, operations manager	Product/solution manager, network manager, project manager
Operating Core	The people who “produce the products and render the services.”	Line worker, market analyst	Software developer, infrastructure engineer, systems operator
Technostructure	Those who standardize the work of the Operating Core and Middle Line, interpreting the desires of the Strategic Apex	Strategic planning, quality assurance	Enterprise architecture, IT Service Management capability, Project Management Office
Support staff	Those who enable the work of the line and other staff.	Corporate legal, human resources, purchasing, corporate finance	IT Asset Management, IT Vendor Management

Mintzberg further developed his theory by proposing that each of these five areas exerted a characteristic “pull” on the organization:



Adapted from [Mintzberg1983]

Definitions and examples for business and IT:

Pull	Definition	Mainstream example	IT example
To Centralize (Strategic Apex)	Tendency to control all decision making	CEO presiding over functional executives (Sales, Marketing, Finance, Operations, IT)	CIO presiding over activity-centric development, engineering/operations, security, architecture silos, very project and ticketing driven
To Balkanize (Middle Line)	Tendency to pull power from both top and bottom, ultimately resulting in divisionalized form	Divisionalized corporation or more granular product/LOB-centric organization.	Dedicated product and infrastructure organizations by line.
To Professionalize (Operating Core)	Tendency to minimize control of administrators and technostucture, seeking independence through increasing sophistication of skills and tacit understandings.	Machine operators seeking to become engineers.	Developers seeking to become software engineers or solutions architects. Desire for certification.

Pull	Definition	Mainstream example	IT example
To Standardize (Technostructure)	Tendency to make work consistent. Directly opposed to Operating Core's desire to professionalize.	Human Resources standardizing hiring processes. CFO Office standardizing travel policies.	IT Service Managers standardizing ITIL processes. Project Management Offices standardizing project governance. Architects standardizing approved products.

Product and function

Even where they are not part of a value stream, activity-oriented teams tend to standardize their operations over time. Their appetite for offering custom solutions begins to diminish. Complaints begin to surface—“They threw the rule book at us,” “What bureaucracy!”

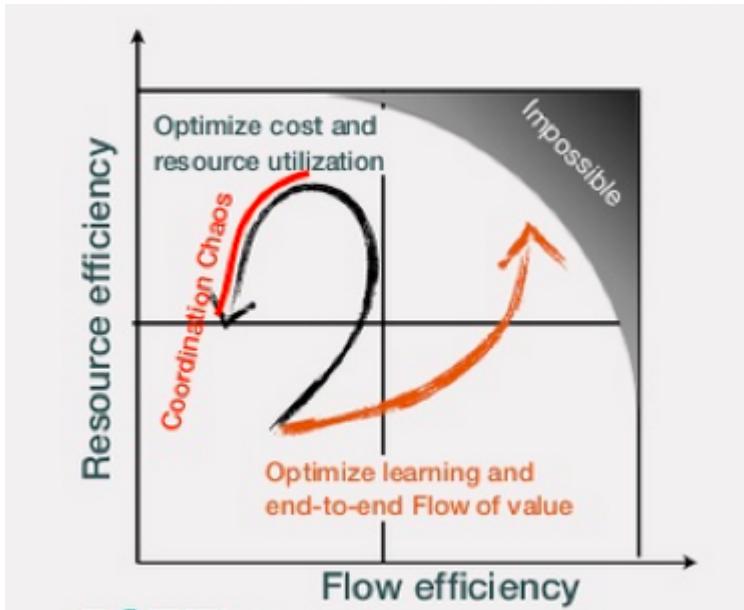
— Sriram Narayan, Agile IT Organization Design

By this time, you probably detect that there is a fundamental tension between functional specialization and end to end value delivery. This tension is the topic of endless research, articles, books, and commentary in the mainstream business press and more specialized channels. The tension may go by different names:

- Flow vs resource
- Process vs silo
- Line vs staff (which has some additional connotations)

Efficiency versus flow

Here is a depiction of one dysfunctional pattern, from an interesting presentation by Ari Tikka [[Tikka2013](#)], that can emerge in navigating this tension:



The above slide is worth a careful examination. The idea is that the more effective you are at product flow, the further to the right you are. The more efficient you are with your resources, the further up you are. It is easy to be completely efficient with resources, and have no flow. It is also possible to be very good at flow, but at the cost of little resource efficiency. Being excellent at both, however, is impossible - there is an outer boundary.

Also, there is an interesting dynamic if you seek too much resource efficiency. Loading staff more and more heavily has counterproductive results, in which flow actually decreases, and the resulting "coordination chaos" also decreases resource efficiency in a vicious reinforcing feedback loop.

Examining the problem more deeply

Complexity responds to competence, not authority.

— Steve Denning

Graphics such as the previous one are limited in insight, however. What are the particular organizational forms and practices in question? We will spend some time thinking through this.

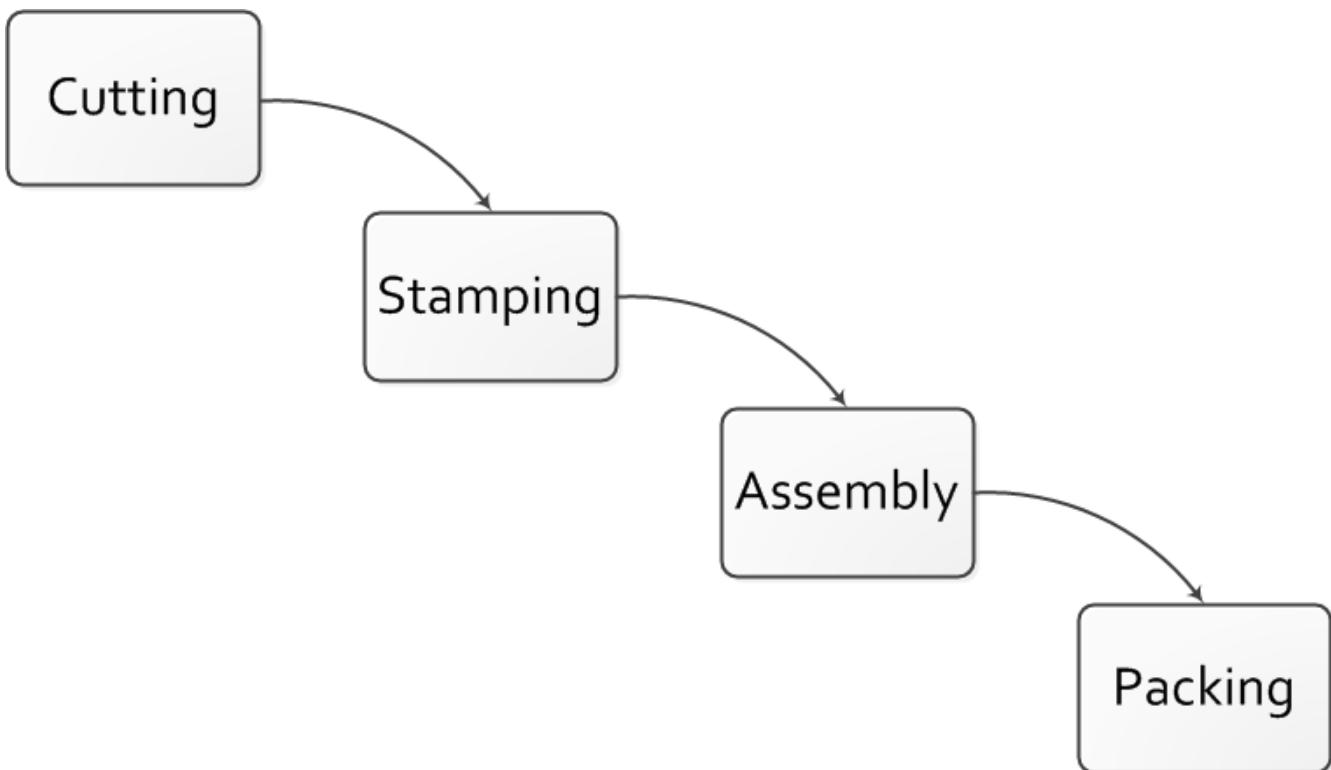
As we saw [previously](#), there are three major concepts used to achieve end to end flow across functional areas:

- Process
- Project
- Product

NOTE

We will go into detail on project management in chapter 8 and process management in chapter 9. But we need to understand how they influence organizational forms.

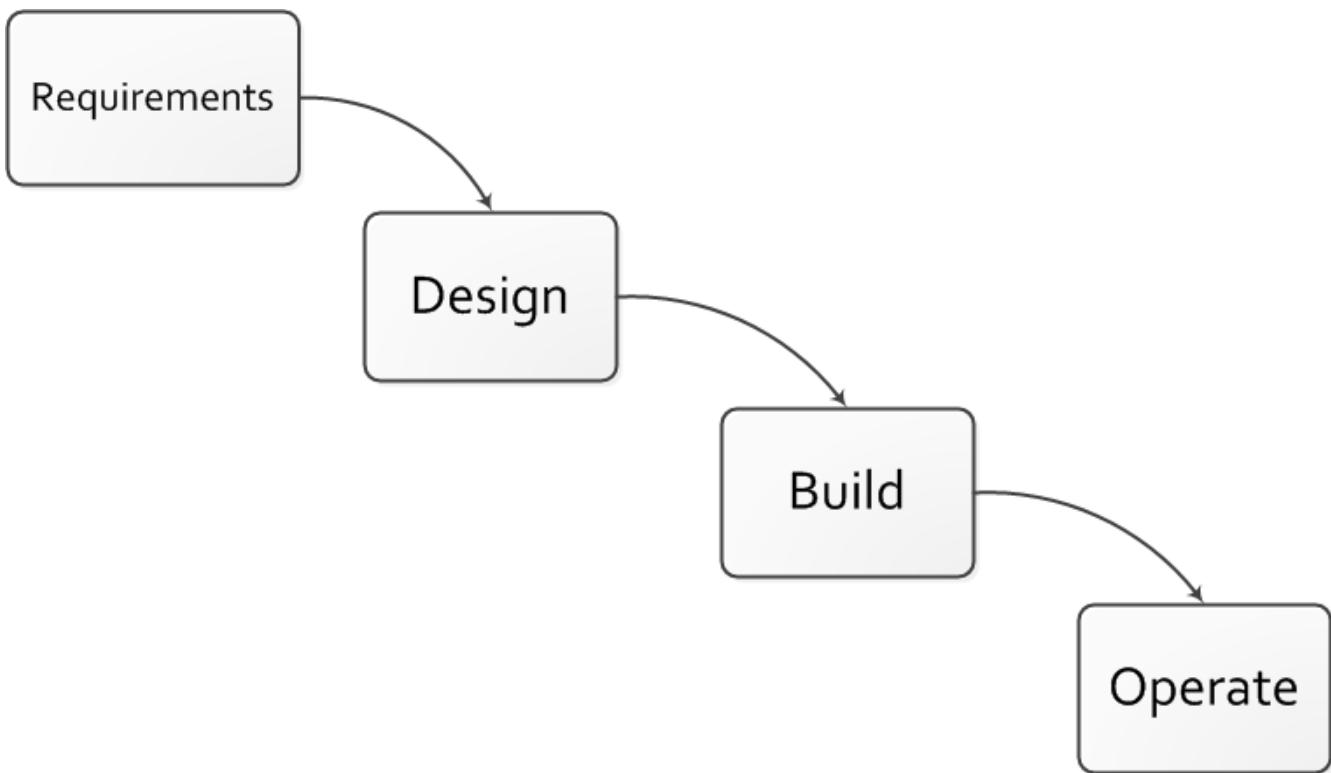
For example, some manufacturing can be represented as a very simple, sequential process model:



The product is already defined and the need to generate information (i.e. through feedback) is at an absolute minimum. Functional areas can continually increase their efficiency and economies of scale through deep specialization.

NOTE Even in this simplest model, feedback is important. Much of the evolution of 20th century manufacturing has been in challenging this naive, open-loop model. (Remember our brief discussion of [open-loop](#)?)

The original, [open-loop](#) waterfall model of IT systems implementation was arguably based on just such a naive concept:



(Review [chapter 3](#) on Agile history.)

There are two primary disadvantages to this model:

- It discourages closed-loop feedback
- There is transactional friction at each handoff

The model falls into what we characterized as the "original sin" of IT management: [confusing production with product development](#). As a repeatable production model, it may work, assuming that there is little or no information left to generate regarding the production process (an increasingly questionable assumption in and of itself).

But when applied to product development, where the **primary goal** is the generation of information, the model is inappropriate and has led to innumerable failures. This includes software development, and even implementing purchased packages in complex environments.

The continuum of organizational forms

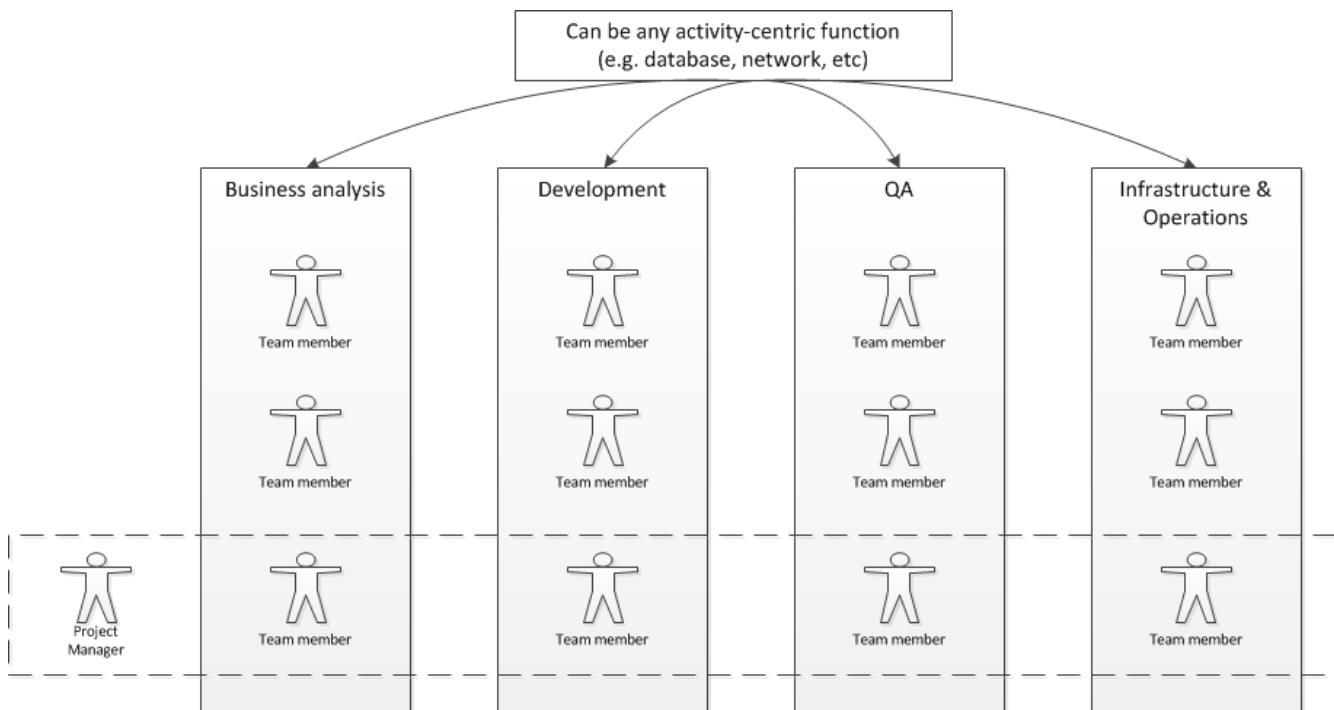
The traditional approach is that we have work and we bring "the right people" to the work. To be nimble—to have organizational agility—we have to have great teams of people and bring the work to the team.

— Joseph Flahiff

The following discussion and accompanying set of diagrams is derived from Preston Smith and Don Reinertsen's thought regarding this problem in *Developing Products in Half the Time* [[Smith1998](#)] and *Managing the Design Factory*. [[Reinertsen1997](#)]. Similar themes are found in the *Guide to the Project Management Body of Knowledge* ([\[PMI2013\]](#)).

There is a spectrum of alternatives in structuring organizations for flow across functional concerns.

First, a lightweight project structure may be implemented, in which the project manager has limited power to influence the activity-based work, where people sit, etc. The project manager is little more than a factory floor expediter in this model.

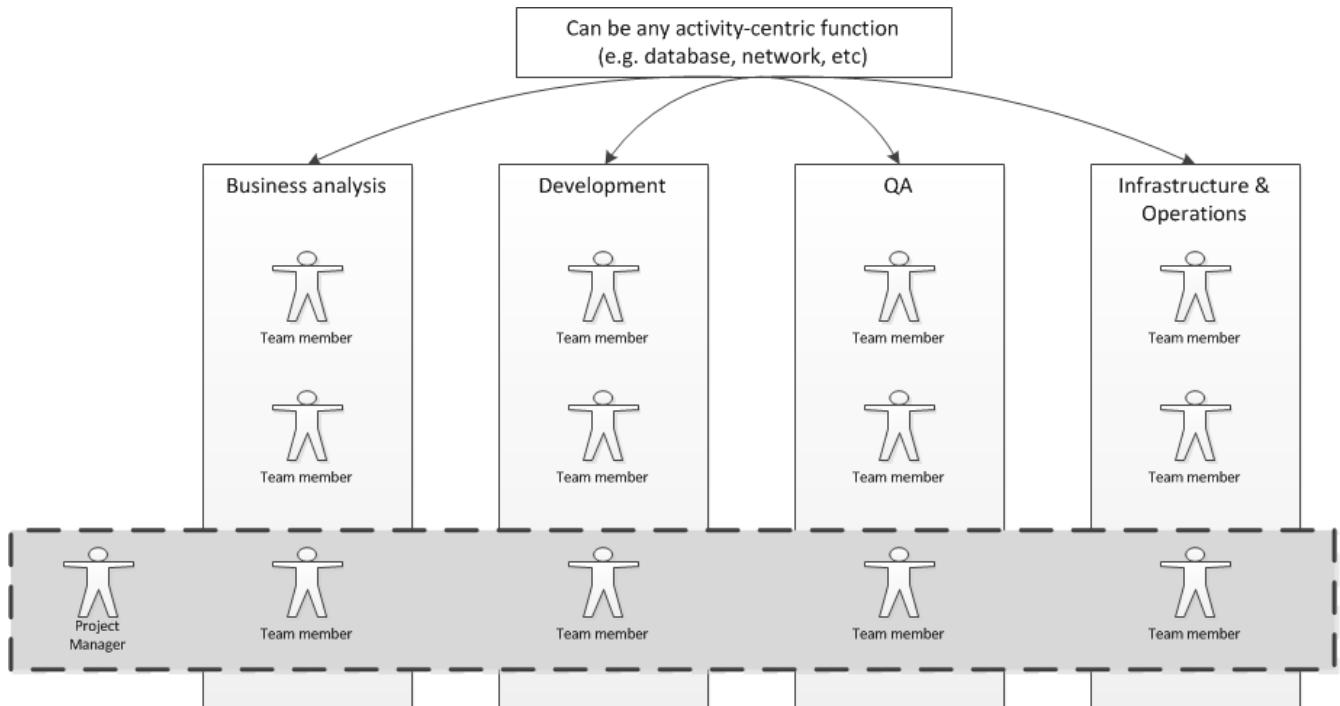


It is in the above model that the following Reinertsen quote is most applicable:

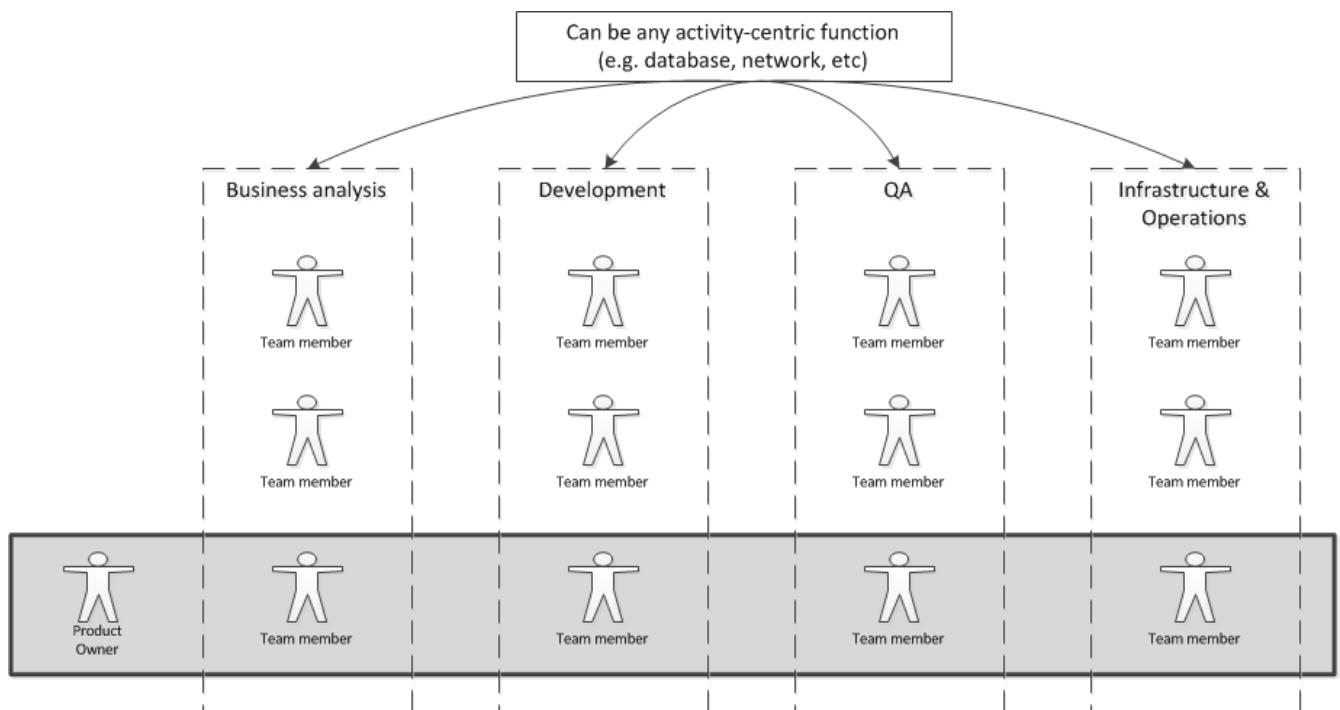
"The danger in using specialists lies in their low involvement in individual projects and the multitude of tasks competing for their time."

Work flows across the functions, perhaps called "centers of excellence," and there may be contention for resources within each center. Often, simple "first in, first out" approaches are used to manage the work, rather than more sophisticated approaches such as Cost of Delay. We will focus in Chapter 9 on approaches to mitigate these issues while retaining this overall organizational form.

Second, a heavyweight project structure may specify much more, including modes of work, standards, etc. The functional manager may be little more than a resource manager, but does still have reporting authority over the team member and crucially still writes their annual performance evaluation (if the organization still uses those.)



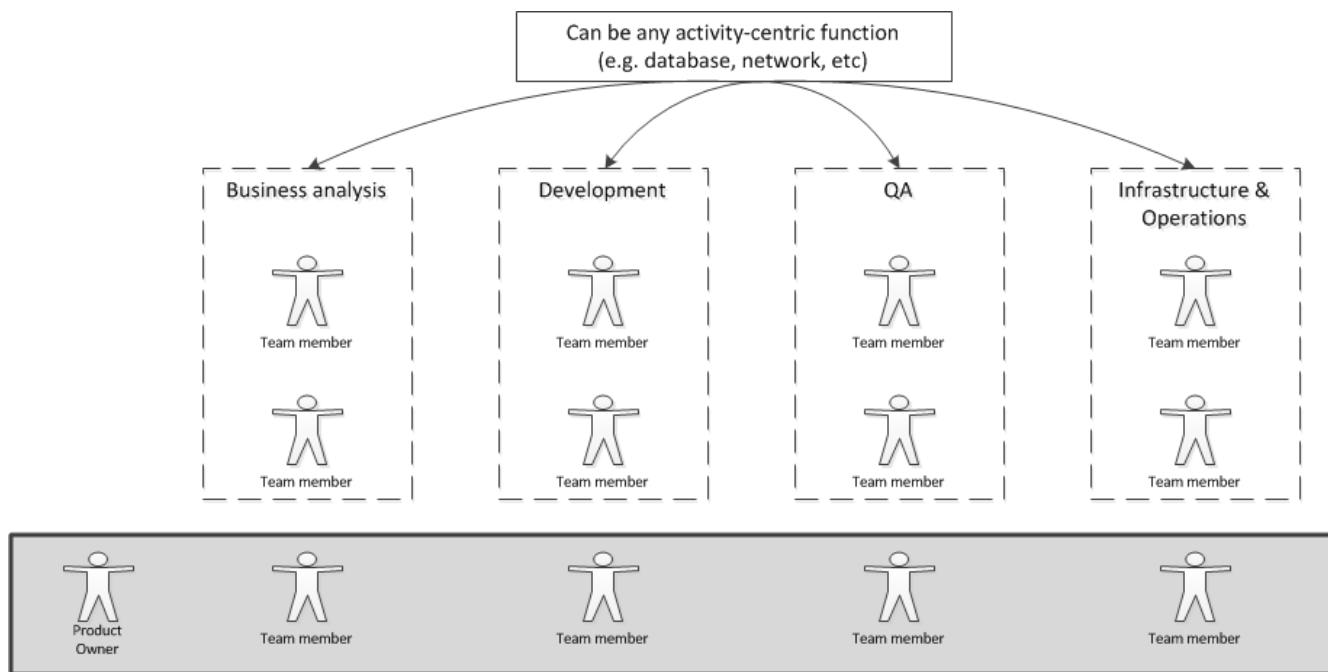
If even more focus is needed—the now-minimized influence of the functional areas is still deemed too strong—the organization may move to completely product-based reporting. With this, the team member reports to the product owner. There may still be communities of interest (the Spotify guilds and tribes are good examples) and there still may be standards for technical choices:



The original Skunk Works [was] established by Kelly Johnson at Lockheed ... Skunkworks have been tried by many others, with outcomes ranging from very good to very bad, and this has led to a variety of opinions on skunk works.

— Preston Smith and Don Reinertsen, *Developing Products in Half the Time*

Finally, in the skunkworks model, all functional influence is deliberately blocked, as distracting or positively injurious to the product vision:



The product team has complete autonomy, and can move at great speed. It is also free to:

- re-invent the wheel, developing new solutions to old and well-understood problems
- bring in new components on a whim (regardless of whether they are truly necessary) adding to sourcing and long-term support complexity,
- ignore safety and security standards, resulting in risk and expensive retrofits.

Ultimately, skunkworks is not scalable. Research by the Corporate Executive Board suggests that "Once more than about 15% of projects go through the fast [skunkworks] team, productivity starts to fall away dramatically." It also causes issues with morale, as a two-tier organization starts to emerge with elite and non-elite segments [\[Goodwin2015\]](#).

A skunkworks team is like a long lived branch. When it comes back in, things are going to break.

— Kevin Matheny

Because of these issues, Don Reinertsen ([\[Reinertsen1997\]](#)) observes that:

Companies that experiment with autonomous teams learn their lessons, and conclude that the disadvantages are significant. Then they try to combine the advantages of the functional form with those of the autonomous team.

— Don Reinertsen, Managing the Design Factory

The Agile movement is an important correction to dominant IT management approaches employing [open-loop](#) delivery across centralized functional centers of excellence. However, the ultimate extreme of the skunkworks approach cannot be the basis for organization across the enterprise.

We will examine the various adaptations and approaches for balancing the two organizational extremes further in Chapters 8 (Project Management) and 9 (Process Management).

Culture

What is culture in this context? It is not so much about an informal dress code, flexible hours, or a free in-house cafeteria as it is about how decisions are taken, norms of behavior, protocols of communication, and the ways of navigating hierarchy and bureaucracy to get things done.

— Sriram Narayan, Agile IT Organization Design

When we look at a particular attribute of corporate culture, we can ask what aspect of the environment or what organizational goals caused that attribute to develop and question why it is a good solution to that environmental challenge. When we look at a rule or a piece of bureaucratic machinery, we might just as well ask what problem it was intended to solve and why it was considered to be a good solution.

— Mark Schwartz, The Art of Business Value

"Culture" is a difficult term to define, and even more difficult to characterize across large organizations. It starts with how an organization is formally structured, as structure is in part a set of expectations around how information flows. "Who talks to who, when and why" is in a sense culture. Culture can also be seen embedded in artifacts like processes and formally specified operating models. Certainly, much in the previous section on organizational theory implied culture; Mintzberg's Five Forces all come with their own cultural tendencies.

But "culture" has additional, less tangible meanings. The anecdotes executives choose to repeat are culture. Whether an organization tacitly condones being 5 minutes late for meetings (because walk time in large facilities is expected) or has little tolerance for this (because most people dial in) is culture. The degree of deference shown to senior executives, and their opinions, is culture. Whether a junior person dares to hit "reply-all" on an email including her boss's boss is culture.

Culture cannot be directly changed - it is better seen as a lagging indicator, that changes in response to specific practical interventions. Even tools and processes can change culture, if they are judiciously chosen (and most tools and processes do **not** have this effect). Skeptical? Consider the impact that computers - a tool - have had on culture. Or email.

We'll look at culture from a few different perspectives in this section:

- Motivation
- Schneider matrix
- Mike Rother's research into Toyota's improvement and coaching "katas"

- Emotional intelligence
- Paul Glen's work on "Leading Geeks"

Basic ingredients & factors

- communication norms
- decision making norms
- conflict resolution norms
 - causes
 - facts
 - methods
 - goals
 - values
 - resolution mech
 - competition
 - collaboration
 - accommodation

getting to yes, etc - conflict resolution processes/lit

forming/storming/norming/performing

Need for longer lived teams <http://searchcio.techtarget.com/tip/How-organizational-agility-will-save-and-destroy-your-company>

Motivation

One of the most important reasons to be concerned for culture is its effect on motivation. There is little doubt that a more motivated team performs better than an unmotivated, "going through the motions" organization.

In general, it is an article of faith in the Agile and Lean communities that motivated teams are more likely to collaborate effectively and seek creative, novel solutions to problems. (I am not sure if there is evidence to this effect, but it seems intuitively obvious.)

But what motivates people?

One of the oldest discussions of culture is Douglas McGregor's idea of **Theory X** organizations, which he developed in the 1960s at the Massachusetts Institute of Technology.

"Theory X" organizations rely on extrinsic motivators and operate on the assumption that workers must be cajoled and punished in order to produce results. We see Theory X approaches when organizations focus on pay scales, bonuses, titles, awards, writeups/demerits, performance appraisals, and the like.

Theory Y organizations operate on the assumption that most people seek meaningful work intrinsically and that they have the ability to solve problems in creative ways that do not require tight standardization. According to Theory Y, people can be trusted and should be treated as mature individuals, in contrast to the distrust inherent in Theory X.

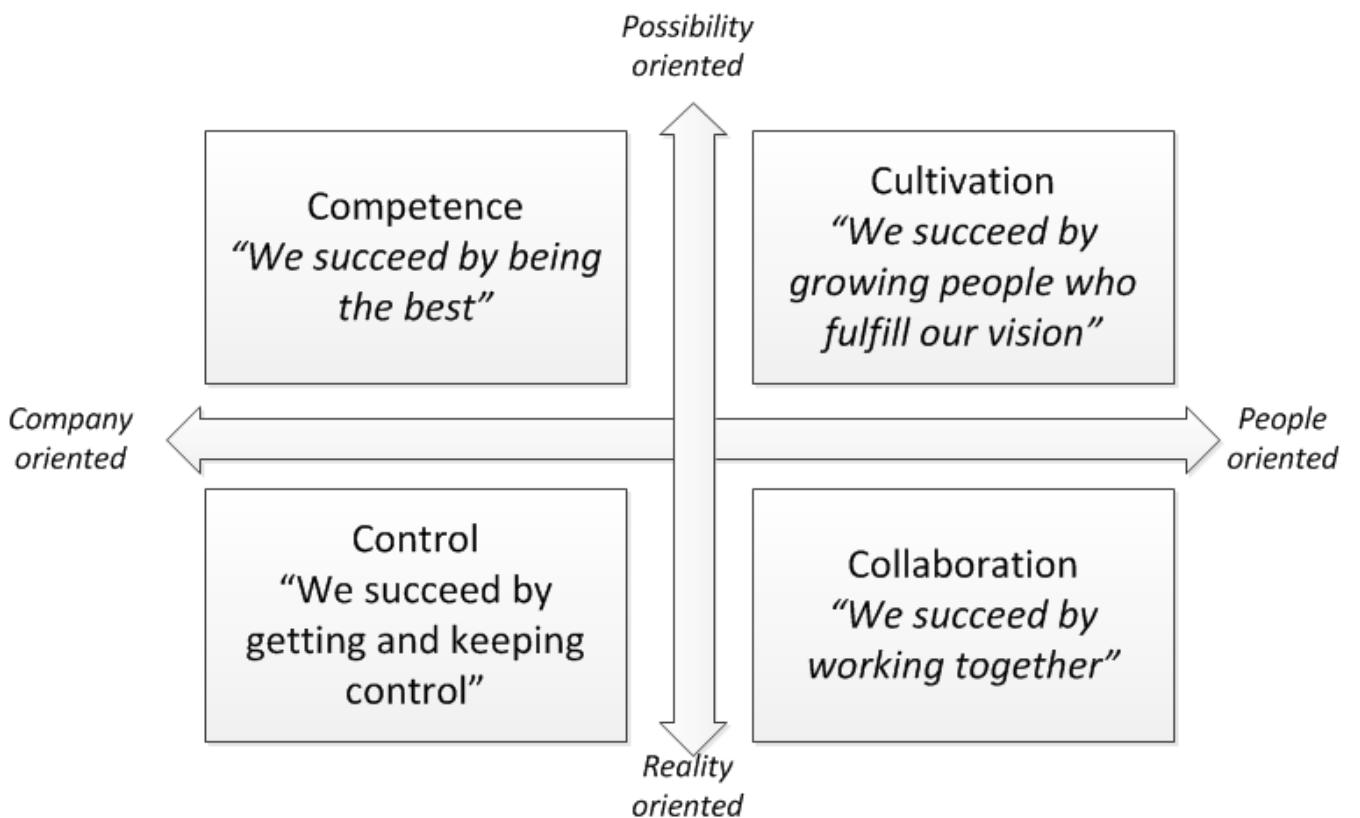
IMPORTANT This book is biased towards Theory Y.

In terms of intrinsic motivation, Daniel Pink, author of Drive, suggests that three concepts are key: autonomy, mastery, and purpose. If these three qualities are experienced by individuals and teams, they will be more likely to feel motivated and collaborate more effectively.

[More on A, M, P]
 [Failure of material incentives]

Schneider matrix, and the failure of control culture

One model for understanding culture is the following matrix [proposed by Schneider](#):



Two dimensions are proposed:

- the extent to which the culture is focused on the company or the individual
- the extent to which the company is "possibility-oriented" versus "reality-oriented"

This is not a neutral matrix. It's not clear that highly controlling cultures are ever truly effective. Even in the military, which is generally assumed to be the ultimate "command and control" culture,

there are notable case studies of increased performance when more empowering approaches were encouraged.

For example, in *Turn the Ship Around: A True Story of Turning Followers Into Leaders*, Captain L. David Marquette discusses moving from a command-driven to an outcome-driven model and the beneficial results it had on the *USS Santa Fe*. Similar themes appear in Captain D. Michael Abrashoff's *It's Your Ship: Management Techniques from the Best Damn Ship in the Navy*.

Neither of these accounts is surprising, when one considers the more sophisticated aspects of military doctrine. Don Reinertsen provides a rigorous overview in chapter 9 of *Principles of Product Development Flow*. In this discussion, he notes that the military has been experimenting with centralized vs. decentralized control for centuries. Modern warfighting relies on autonomous, self-directed teams that may be out of touch with central command and required to improvise effectively to achieve the mission. Therefore, military orders are incomplete without a statement of "commander's intent" - the ultimate outcome of the mission. [Reinertsen2009], pp. 243-265.

Toyota Kata

Six years ago I began the research that led to [Toyota Kata] thinking, like just about everyone else, that the story was about techniques and other listable aspects of Toyota. Today I see Toyota in a notably different light: as an organization defined primarily by the unique behavior routines it continually teaches to all its members.

— Mike Rother, Toyota Kata

Academics and consultants have been studying Toyota for many years. The performance and influence of the Japanese automaker is legendary, but it has been difficult to understand why.

Much has been written about Toyota's use of particular tools, such as kanban bins and andon boards. However, Toyota views these as ephemeral adaptations to the demands of its business.

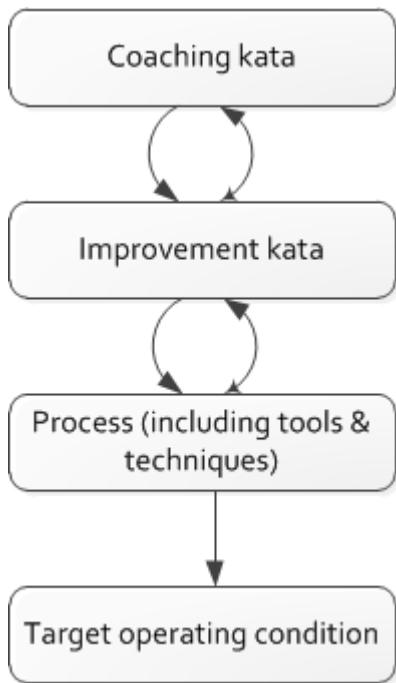
According to Mike Rother in *Toyota Kata*, underlying Toyota's particular tools and techniques are two powerful practices:

- The improvement kata
- The coaching kata

What is a *kata*? It is a Japanese word stemming from the martial arts, meaning pattern, routine, or drill. More deeply, it means "a way of keeping two things in alignment with each other."

The improvement kata is the repeated process by which Toyota managers investigate and resolve problems, in a hands-on, fact-based, and preconception-free manner, and improve processes towards a "target operating condition."

The coaching kata is how the improvement kata is instilled in new generations of Toyota managers.



As Rother describes it, the coaching and improvement katas establish and reinforce a coherent culture or mental model of how goals are achieved and problems approached. It is understood that human judgement is not accurate or impartial. The method compensates with a teaching-by-example focus on seeking facts without preconceived notions, through direct, hands-on investigation and experimental approaches.

This is not something that can be formalized into a simple checklist or process; it requires many guided examples and applications before the approach becomes ingrained in the upcoming manager.

Open and closed loop control in culture

We can understand Toyota Kata partly in terms of systems theory (Section II) - the katas are in a sense closed-loop control, working on several levels in a layered fashion. [link]

(There are interesting theories and models that thinkers such as Stafford Beer have proposed for understanding systems of closed-loop control - Viable Systems Model.)

Conjecture: "command and control culture" means open loop control.

Notes

Liker quadrant from Toyota Way (cited by Cohn)
Westrum typology (State of DevOps)

Agile coaching and culture
Lyssa Adkins

The role of external facilitators

Google research on high trust cultures

http://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html?_r=0

Emotional intelligence

- Culture != 'collaborate & be nice' - but intelligent IT pros do sometimes have issues - autism spectrum/Aspergers, empathy, etc ...

(to be written)

"Flow" and the individual

Summarize <<Csikszentmihalyi1990>>

"Leading Geeks"

Because power is about the regulation of behavior, it has very little effect on creativity. Traditional methods of exercising control have little positive effect on the inner state of mind of geeks.

— Paul Glen, Leading Geeks

(to be written)

Teaming (Amy Edmondson)

The problem of culture "change"

[to be written]

effective practices: traditions, cadence

Basics of professional conduct

Authority, responsibility, accountability Delegation, commitment Micromanagement.

(examples) Mission statements...

incorporate https://en.wikipedia.org/wiki/Edgar_Schein

5 Dysfunctions of a team

Topics in organization and culture

Collaborative

_ 20160609 Status: Like most topics sections, this one will remain rough until 2nd draft. Suggestions appreciated._

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

The mainstreaming of IT into the business

"Rogue" IT

Convergence diagram.

ad-hoc & governance structures, committees

Communities of practice originated by Wenger, McDermott, and Snyder 2002.

Types of COPs

Feature vs Component at scale

We have previously discussed [feature vs component teams](#). As a reminder, features are functional

aspects of software (things people find directly valuable) while components are how software is organized (e.g. shared services and platforms such as data management).

As an organization grows, we see both the feature and component sides scale. Feature teams start to diverge into multiple products, while component teams continue to grow in the guise of shared services and platforms. Their concerns continue to differentiate and communication friction may start to emerge between the teams. How an organization handles this is critical. (forward ref to chap 9 shared svcs, section IV lifecycles)

Adam Smith and Frederick Taylor

- Taylor & cultural antipatterns (including the 'you just don't get X' antipattern)

Bi-modal IT, a critique

[to be written]

reference [How CIOs can raise their *IT clock speed* as pressure to innovate grows](#)

Solving for decisionmaking & governance

No choice but to allocate fine-grained decision rights between product lines & guilds.

See Reinertsen/Smith.

Org change management -

ProSci/ADKAR, Lean Change Mgmt

the limits to "you build it, you run it" - employees are not owners...

culture leads, compliance lags

The problem of epistemic closure

additional material

Toyota Way quadrant as cited in <>Cohn2010>> p145

IT and learning (Langer)

The business within a business paradigm has failed

Microservices.... important - how multiple services interact.

The rise of coaching

incentives - making across the board incentives work, Knez 2002, harvard biz review

Etsy : designated vs dedicated - Kevin M discussion - mentioned there was talk @ DevOpsDays MSP 2015

Training Within Industry

Github abandoning flat structure <http://www.bloomberg.com/news/articles/2016-09-06/why-github-finally-abandoned-its-bossless-workplace>

Connor:Discussing Design: Improving Communication and Collaboration through Critique

Second draft

include <http://www.slideshare.net/meetfelipe/slides-agile2016-stop-using-agile-with-waterfall-goals-goal-agility-with-okr>
https://www.amazon.com/dp/1119252393/ref=cm_sw_r_tw_awdo_x_YQKZxb3GM0P4P

think more about whether OKR belongs section II or III

allspaw on joint activity. requires common ground of understanding and a compact of joint action http://paperswelove.org/2016/video/john-allspaw-common-ground/?utm_content=buffer893c4&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

Conclusion

Beware of simplistic progression models, which imply that organizational forms evolve from authoritarian to enlightened. You never get to done with organizational development and

evolution. It is always an ongoing process of balancing culture, centralization, and autonomy. Rather than seeking any "best practice," it is better to see a toolbox, or perhaps a set of levers you can position in various ways to get different results.

As of 2015, product-centric Agile teams are becoming more and more common. But there are ultimately limitations to this approach, which many companies have experienced.

Culture can be difficult to define. We saw how Toyota's success can be understood as a strategy of ingraining particular cultural understandings through pervasive coaching.

Discussion questions

- What kinds of organizational structures have you worked in?
- What do you prefer?

Research & practice

- Review the history of the original Lockheed Martin Skunkworks under Kelly Martin.
- As a class, group yourself into the various organizational forms and route simple paper-based work between each other, or undertake simple craft projects such as building a Lego set as a team.

Further reading

author's note: review Team of Teams

Books

- Henry Mintzberg, [Structure in Fives](#)
- Sriram Narayam, [Agile IT Organization Design](#)
- Tom DeMarco, Timothy Lister, [Peopleware](#)
- Fred Brooks, [The Mythical Man-Month](#)
- Paul Glen, [Leading Geeks](#)
- Daniel Pink, [Drive](#)
- Preston G. Smith, Donald G. Reinertsen, [Developing Products in Half the Time: New Rules, New Tools](#)

Articles

- Melvin Conway, [How Do Committees Invent?](#) (original statement of Conway's Law)
- [The Misalignment of Product Architecture and Organizational Structure in Complex Product Development](#) (Research supporting Conway's Law)
- [The Marriage of Agile and Organizational Change Management](#)
- [How CIOs can raise their *IT clock speed* as pressure to innovate grows](#)

- [11 Ways to Motivate Geeks](#)
- [Enough of silos? How about slabs?](#)
- [WORKER, INTERRUPTED: THE COST OF TASK SWITCHING](#)
- [DevOps - Threat or Menace?](#)
- [How organizational agility will save and destroy your company](#)
- [What Google Learned From Its Quest to Build the Perfect Team](#)

Videos

- [Spotify model, part 1](#)
- [Spotify model, part 2](#)

Other resources

In addition to the above books, the resources available on organizational design at smaller scales are vast, as these are very frequently asked questions by entrepreneurs of all types. Some useful links are:

- <http://www.ehow.com>
- <http://www.entrepreneurship.org>

Chapter 8: Project & Resource Management

thinking about renaming to "Investment and Execution" - project management continues to lose favor...

budgeting vs investing

Introduction to Chapter 8

A project is a temporary endeavor undertaken to create a unique product, service, or result.

— Project Management Body of Knowledge, version 5

We might restate the above quote as **the execution of a given scope of work within constraints of time and budget**. But questions arise. We are already executing work, without this concept of "project." We have discussed [Scrum](#) in Chapter 4, [Kanban](#) in Chapter 5, and various organizational forms for performing work in [Chapter 7](#).

And what is this idea of "scope"? How would we know it in advance, so that the "constraints of time and budget" are reasonable?

As we have seen in our discussions of [Product Management](#), in implementing **truly new products**,

(including digital products) estimating time and budget is virtually impossible because **the necessary information is not available**. In fact, creating it - which requires tight feedback loops - is the actual work of the "project". Therefore, in the new Agile world, there is some uncertainty as to the role of and even need for traditional project management. ([\[Cobb2015\]](#) is a thorough examination of these issues.)

In the project management literature and tradition, much focus is given to the **execution** aspect of project management - its ability to help manage complex, interdependent work across resource limitations. However, we are also interested in the **structural role** of project management. Project management may be institutionalized through establishing an organizational function known as the Project Management Office (PMO), which may use a concept of project portfolio as a means of constraining and managing the organization's multiple priorities. What is the relationship of the traditional PMO to the new, product-centric, digital world?

At a practical level, project managers remain important roles and (sometimes re-labeled Scrum masters) are employed in both traditional and new digital companies as overall orchestrators of complex initiatives. This chapter will examine some of the reasons for project management's persistence and how it is adapting to the new product-centric reality.

IMPORTANT

There is little doubt the student pursuing an IT career will encounter project managers and their techniques at some point. This chapter will discuss the fundamental premises, techniques, and limitations of project management, in the context of digital transformation.

Financial planning and resource management have not gone away, and without project management financial planning meets challenges, especially regarding the efforts most critical to the business. The very concept of having a vision becomes problematic without some ability to plan. And the role of external vendors expands to a point require much more formalized approaches. Because of these dependencies, this chapter also covers IT financial management, sourcing, and human resource management at an introductory level.

Chapter 8 outline

- Why project management?
 - basic concerns and disciplines
 - PMBOK etc (also covered in Appendix)
 - PM relationships w/IT & Agile
 - Impact of Cloud on project management
- Project mgmt theory -
 - Critical chain
 - Execution, resource & scheduling constraints
 - Last responsible moment
- Project vs product management
 - PM still needed in an Agile, product-centric world?

- Project management antipatterns
- The estimation controversy (sidebar?)
- The future of project management
- Sourcing
 - Basic concerns
 - Impact of Cloud
- Resource management
 - Basic concerns
 - Impact of Agile
 - Time tracking
 - SFIA framework
- IT financial management
 - Basic concerns
 - Agile challenges (Lean Accounting, Beyond Budgeting)
- Topics
 - 2 dimensions of demand management
 - Frameworks
 - What is a framework?
 - PMBOK
 - Frameworks for Agile at scale

Chapter 8 learning objectives

- Identify the basic practices of project management
- Understand fundamental concepts of project & execution theory
- Identify the basic concerns of IT sourcing
- Identify the basic concerns of IT resource management
- Identify the basic concerns of IT financial management

second draft: start w/investment

reconsidering title: investment and delivery

Team is smallest increment of labor, sprint is smallest unit of labor

Why project management?

An ongoing work effort is generally a repetitive process that follows an organization's existing procedures. In contrast, because of the unique nature of projects, there may be uncertainties or differences in the products, services, or results that the project creates.

— Project Management Body of Knowledge, version 5

In our [emergence model](#), we always seek to make clear **why** we need a new concept or practice. It is not sufficient to say, "we need project management because companies of our size use it." Many authoritative books on Agile software development assume that some form of project management is going to be used. Other authors question the need for it, or at least raise various cautions.

Project management, like many other areas of IT practice, is undergoing considerable transformation in response to the Agile transition. However, it will likely remain an important tool for value delivery at various scales.

What is a project?

As noted in the introduction, we are already doing work in our emergence model, using Scrum and Kanban for the most part. But there is interest in more formalized, traditional project management, so some of you go to training and learn the following basics.

IMPORTANT

As with other chapters in the later part of this book, we are going to introduce Project Management "on its own terms." We will then add additional context and critique in subsequent sections.

Let's start with the most fundamental aspects of a project. You can start your understanding of a project by first thinking of a "to-do" list. However, rather than a miscellaneous list of errands, a project is a to-do list that is aimed at delivering some overall end result, with certain resources and within a time frame.

So, comparing to your daily life, a random list of to-dos isn't a project, but if you are organizing your annual holiday party, that certainly would qualify as a project.

We could start with the following list:

Choose menu

Clean house before party

Get groceries

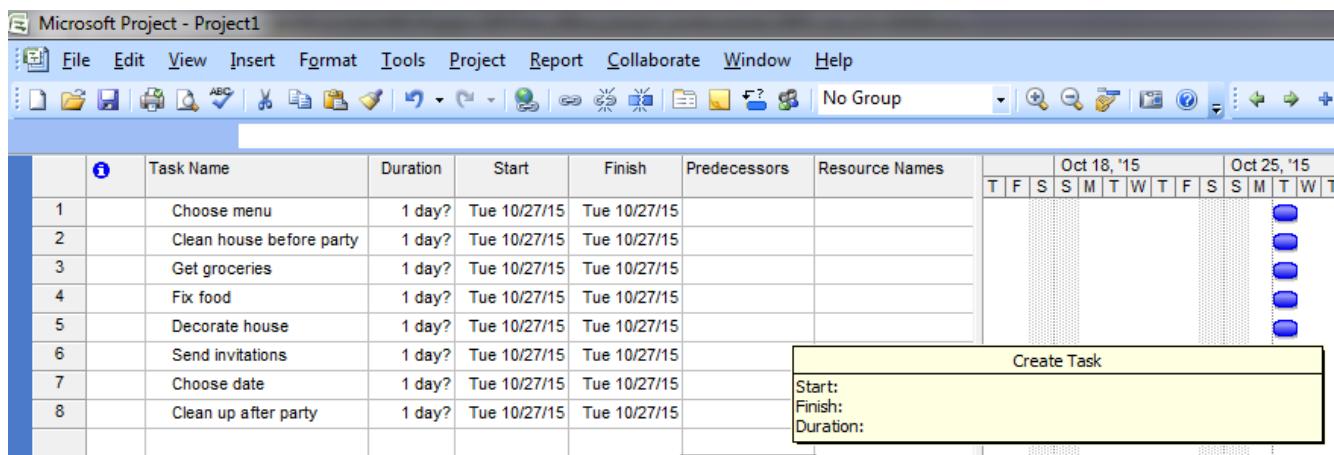
Fix food
Decorate house
Send invitations
Choose date
Clean up after party

One of the important characteristics of a project is that it is temporary. This project ends when we clean up after the party.

Project management, however, takes things much further. There are a number of ways that the list above can be refined using project management techniques.

First, we should probably organize the list a little better. And the task "clean house" - do we understand precisely what we mean by that? Maybe we should break this down into sub-tasks. Also, you may notice that there are "dependencies." For example, we can't send out invitations before we choose the date, nor can we fix the party food before we get groceries, which in turn depends on setting the menu.

Let's break out the heavyweight tools. Introducing Microsoft Project:



First, let's set up some sub-tasks and document the dependencies:



Notice the bold headings:

- Clean house before party

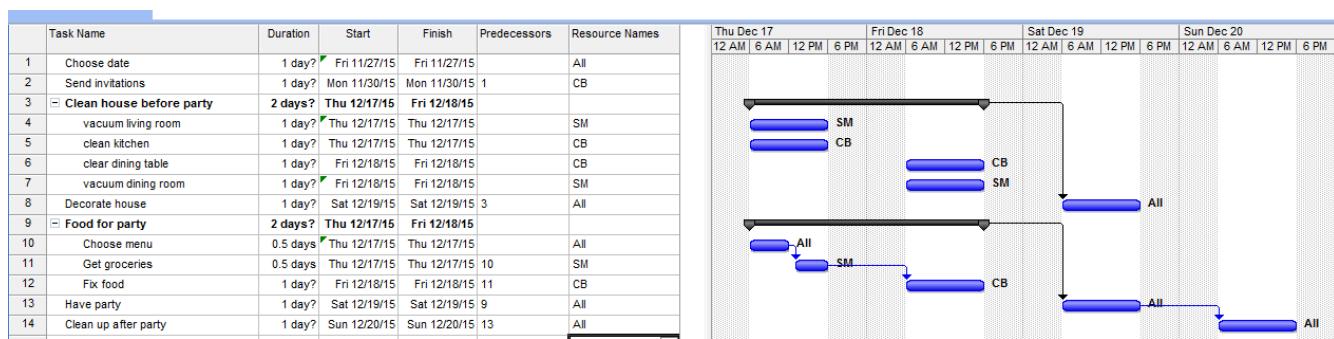
- Food for party

and how they have indented "sub-tasks" underneath them. These are all the things that need to be completed, for the overall major task to be considered complete.

For example, we can't call the house clean if the dining room isn't vacuumed.

Also notice that, to the right, we see certain tasks are linked by thin black arrows. We are not going to decorate the house until all the cleaning is done, and we need to get the food prepared before we have the party. As mentioned above, these are called dependencies.

As we look at the schedule, we see that the time is off. The party will be on December 19, so we need to adjust the dates. In addition, let's indicate the initials of the person who is responsible for a given task.



This is just the tip of the iceberg. Much more can be tracked in a formal project management approach, such as:

- Cost of people's time - forecast and actual
- % complete of a given task
- Which sequence of tasks is the "critical path" - the sequence that must be done in order, that will take the longest time

A project manager is responsible for all of this and more. A critical concept is that of "deliverable." In this example, the deliverable is the outcome of each task:

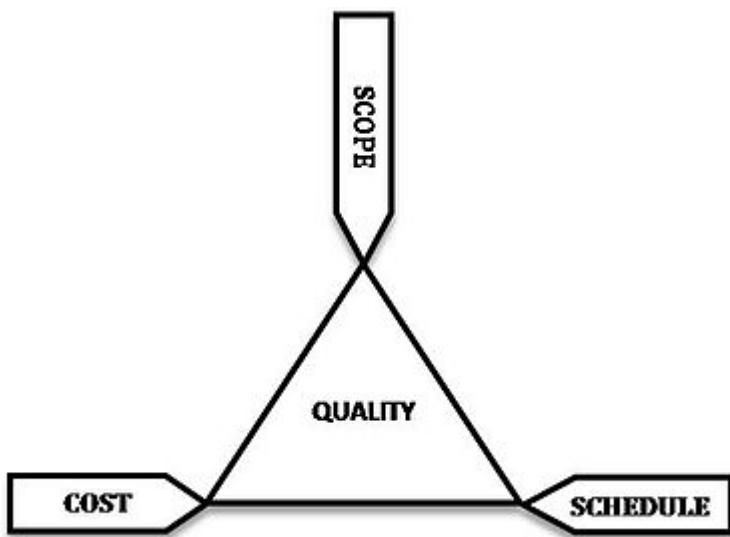
- invitations sent by 11/30 by CB
- menu decided by 12/17 by all
- house cleaned by 12/18 by all
- food prepared for guests by 12/18 by CB

Notice that deliverables are defined by "what," "who," **and** "when."

While it is the responsibility of each contributor to the project to "meet" their deliverables, the project manager checks in regularly with each of them as to whether they will in fact be able to do this. In larger, more complex projects, the project manager updates the project plan with the effort expended to date, and the estimated time to completion. Individual status reports are assembled ("rolled up") into an overall picture of project status, reported to executives with a high level

assessment of whether the project is likely to meet its objectives of cost, scope, quality, and schedule.

The "iron triangle"



Project Management "Iron Triangle" from Wikipedia

The project management [Iron Triangle](#) represents the interaction of cost, time, scope, and quality on a project. The idea is that, in general, one or more of these factors may be a constraint. The following sign is often seen in service organizations:



[photo credit](#)

The same applies to project management and reflects well the "iron triangle" of tradeoffs.

note: we will critique the iron triangle subsequently

How is a project different from simple "work management"?

In [Chapter 5](#), we covered a simple concept of "work management" that deliberately did not differentiate product, project, and/or process-based work. As was noted at the time, for smaller organizations, most or all of the organization would be the "project team," so what would be the point?

The project is starting off as a list of tasks, that is essentially identical to a product backlog. Even in Kanban, we know who is doing what, so what is the difference? Here are key points:

- The project is explicitly time-bound. As a whole, it is lengthier and more flexible than the repetitive, time-boxed sprints of Scrum, but more fixed than the ongoing flow of Kanban.
- Dependencies. You may have had a concept of one task or story blocking another, and perhaps you used a whiteboard to outline more complex sequences of work, but project management has an explicit concept of dependencies in the tasks, and powerful tools to manage them. This is essential in the most ambitious and complex product efforts.
- Project management also has more robust tools for managing people's time and effort, especially as they translate to project funding. While this may be a contentious aspect of project management (see later in this chapter on the [No Estimates](#) controversy), it remains a critical part of management practice in both IT and non-IT domains.

At the end of the day, people expect to be paid for their time, and investors expect to be compensated through the delivery of results. Investment capital only lasts as a function of an organization's "burn rate;" the rate at which the money is consumed for salaries and expenses. Some forecasting of status (whether that of a project, organization, product, program, or what have you) is therefore an essential and unavoidable obligation of management, unless funding is unlimited (a rare situation to say the least.)

Project accounting, at scale, is a deep area with considerable research and theory behind it. In particular, the concept of Earned Value Management is widely used to quantify the performance of a project portfolio (more on this to come).

A traditional information technology project

So, what does all this have to do with information technology? As we have discussed in previous chapters, project management is one of the main tools used to deliver value across specialized skill-based teams.

A "traditional" IT project starts with the "sponsorship" of some executive with authority to request funding. For example, suppose that the VP of Logistics under the Chief Operations Officer believes that a new supply chain system is required. With the sponsorship of the COO, they put in a request (possibly called a "demand request" although this varies by organization) to implement this system. The assumption is that a commercial software package will be acquired and implemented.

The IT department serves as an overall coordinator for this project. The "demand request" in many cases is registered with the enterprise Project Management Office, which may report under the CIO.

NOTE

Why might the Enterprise Project Management office report under the CIO? IT projects in many companies represent the single largest type of internally managed capital expenditure. The other major form of projects, building projects, are usually outsourced to a general contractor.

The project is initiated by establishing a charter, allocating the funding, assigning a project manager, establishing communication channels to stakeholders, and a variety of other activities.

One of the first major activities of the project will be to select the product to be used.

They will help lead the RFI/RFQ process by which vendors are evaluated and selected.

NOTE

RFI stands for [Request for Information](#); RFQ stands for [Request for Quote](#). See the links for definitions.

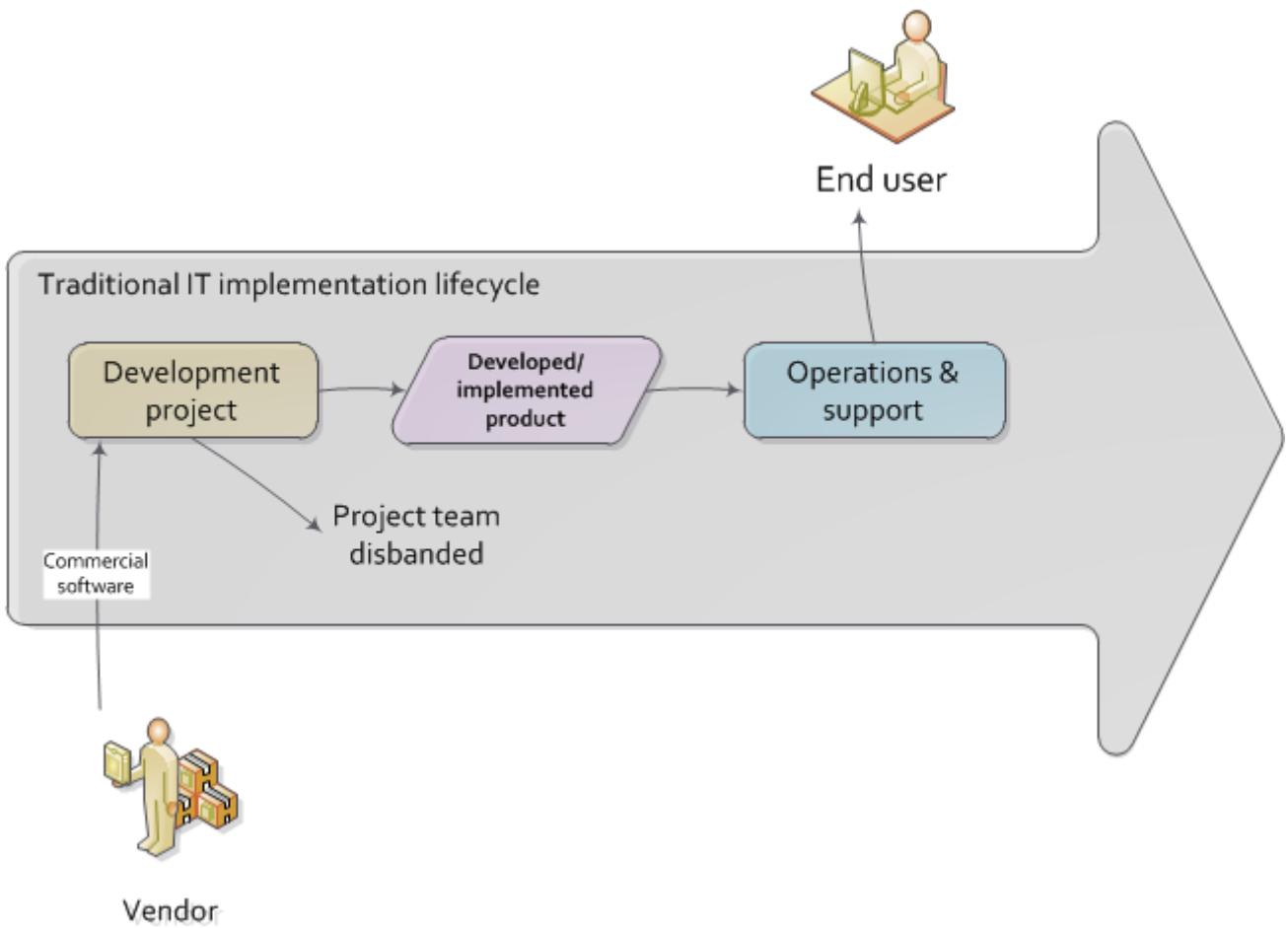
Once the product is chosen, the project must staff up (in reality, staffing arrangements were probably being made at the same time as the RFI/RFQ) and the systems implementation lifecycle can start.

We might call the above, the **systems implementation lifecycle**, not the **software development lifecycle**. This is because most of the hard software development was done by the third party who created the supply chain software. There may be some configuration or customization (adding new fields, screens, reports) but this is lightweight work in comparison to the software engineering required to create a system of this nature.

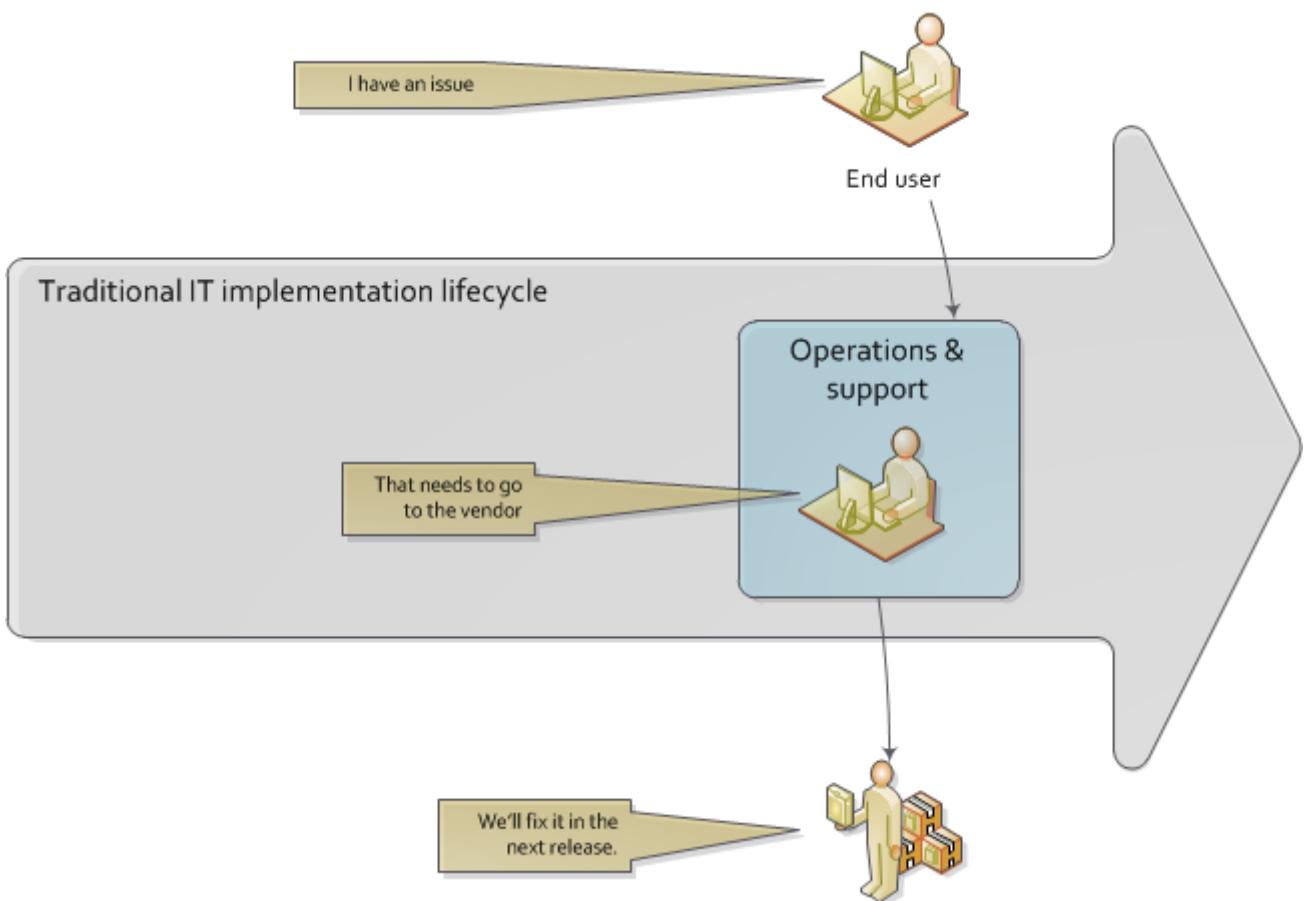
The system requires its own hardware (servers, storage, perhaps a dedicated switch) and specifying this in some detail is required for the purchasing process to start. The capital investment may be hundreds of thousands, or millions of dollars. This in turn requires extensive planning and senior executive approval for the project as a whole.

It would not have been much different for a fully in house developed application, except that more money would have gone to developers. The slow infrastructure supply chain still drove much of the behavior, and correctly "sizing" this infrastructure was a challenge particularly for in-house developed software. (The vendors of commercial software would usually have a better idea of the infrastructure required for a given load.) Hence much attention to up-front planning. Without requirements, no analysis or design; without design, how to know how much server to buy?

Ultimately, the project comes to an end, and the results (if a product such as a digital service) are transitioned to a "production" state. Here is a graphical depiction:



We can see a number of problems with this classic model, starting with the lack of responsiveness to consumer needs:

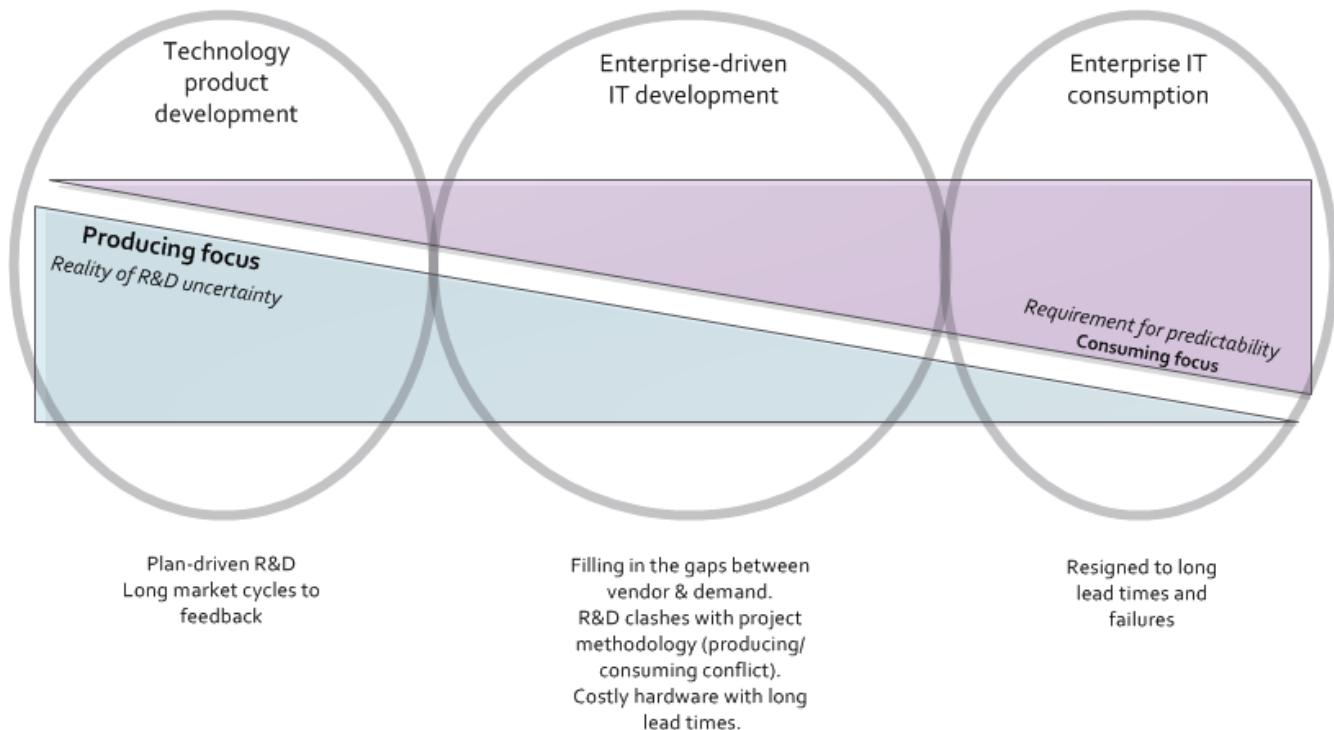


This might be OK for a non-competitive function, but if the "digital service consumer" has other options they may go elsewhere. If they are an internal user within an enterprise, they might be engaged in critical competitive activities.

The decline of the "traditional" IT project

The above scenario is in decline, and along with it a way of life for many "IT" professionals. One primary reason is Cloud, and in particular Software as a Service. Another reason is the increasing adoption of the Lean/Agile product development approach for digital services.

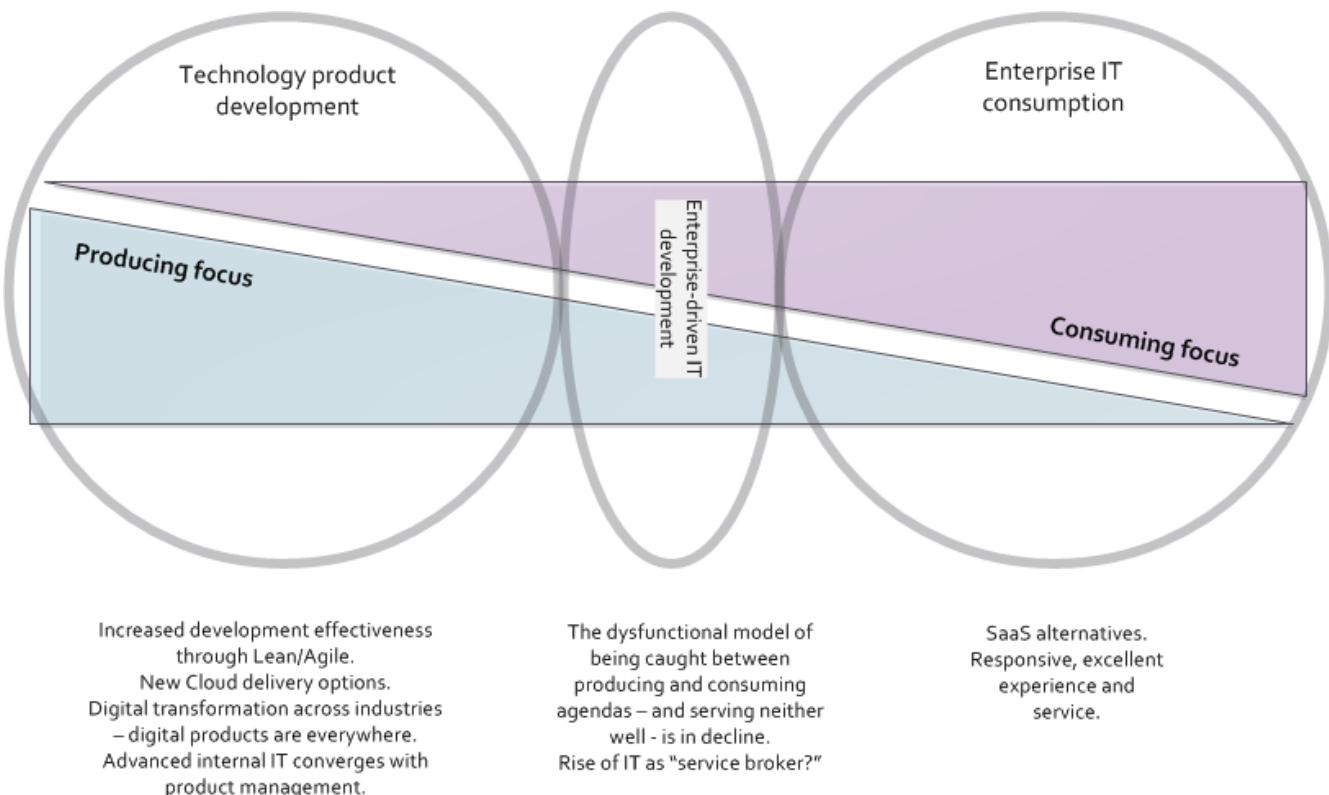
Here is one view of the classic model:



Notice the long triangles labeled "Producing focus" and "Consuming focus." These represent the perspectives of (for example) a software vendor versus their customer. Traditionally, the research and development (R&D) function was most mature in the product companies. What was less well understood was that internal IT development was also a form of R&D.

Because of the desire for scope management (predictability and control), the IT department performing systems development was often trapped in the worst of both worlds - having neither good quality product, nor high levels of certainty. For many years, this was accepted by industry as the best that could be expected.

However, the combination of Lean/Agile and Cloud is changing this:



There is diminishing reason to run commodity software (e.g. payroll, expenses, HR, etc) in-house. Cloud providers such as Workday, Concur, Salesforce, and others provide ready access to the desired functionality "as a service." The responsiveness and excellence of such products is increasing, due to the increased tempo of market feedback (note that while a human resource management system may be commodity for **your** company, it is **strategic** for Workday) and concerns over security and data privacy are rapidly fading.

What is left internal to the enterprise, increasingly, are those initiatives deemed "competitive" or "strategic." Usually, this means that they are going to contribute to a revenue stream. This in turn means they are "products" or significant components of them. (See Chapter 4, [Product Management](#).)

A significant market-facing product initiative (still calling for project management per se) might start with the identification of a large, interrelated set of features, perhaps termed an "epic." Hardware acquisition is a thing of the past, due to either private or public cloud. The team starts with analyzing the overall structure of the epic, decomposing it into stories and features, and organizing them into a logical sequence.

Because capacity is available on demand, new systems do not need to be nearly as precisely "sized," which meant that implementation could commence without as much up front analysis. Simpler architectures suffice until real load is proven. It might then be a scramble to refactor software to take advantage of new capacity, but the overall economic effect is positive, as over-engineering and over-capacity are increasingly avoided.

So, IT moves in two directions - its most forward-looking elements align to the enterprise product management roadmap, while its remaining capabilities may deliver value as a "service broker." (More on this in the section on [IT sourcing](#))

We shall now return to the question of project management in this new world.

Principles and theories of project management

The previous chapter discussed some typical applications of project management, starting with very simple examples and moving into more complex topics, concluding with a discussion of the evolution of IT delivery in the modern enterprises. We return to the specific topic project management with a deeper examination.

Project management (NOT restricted to IT) is a defined area of study, theory, and professional practice. This section provides a (necessarily brief) overview of these topics.

We will first discuss the Project Management Body of Knowledge, which is the leading industry framework in project management, at least in the United States. (PRINCE2 is another framework, originating from the UK, which will not be covered in this edition.) We will spend some time on the critical issues of scope management which drive some of the conflicts seen between traditional project management and Agile product management.

We will compare project execution to more general problems of scheduling and execution, and finally we will take a brief look at Eli Goldratt's Critical Chain.

The Project Management Body of Knowledge

The Project Management Body of Knowledge is sustained by the Project Management Institute. PMI describes itself as:

the world's leading not-for-profit professional membership association for the project, program and portfolio management profession. Founded in 1969, PMI delivers value for more than 2.9 million professionals working in nearly every country in the world through global advocacy, collaboration, education and research. PMI advances careers, improves organizational success and further matures the profession of project management through its globally recognized standards, certifications, resources, tools, academic research, publications, professional development courses, and networking opportunities (from www.pmi.org).

The Project Management Body of Knowledge is articulated in a publication, *A Guide to the Project Management Body of Knowledge*. While this may seem to imply that the PMBOK and its guide are two different things, they are not—it is one publication.

The PMBOK, as of the latest edition, consists of:

- 47 Project Management "processes," grouped into
- 5 Project Management process "groups" and
- 10 Project Management "knowledge areas"

The groups are the easiest to start with. They are:

- Initiating
- Planning
- Executing

- Monitoring and Controlling
- Closing

The PMBOK is clear that the "Process Groups are not project phases. In fact, it is possible that all Process Groups could be conducted within a phase." [\[PMI2013\]](#), A1.3.

The Knowledge Areas are a different dimension, and consist of:

- Project Integration Management
- Project Scope Management
- Project Time Management
- Project Cost Management
- Project Quality Management
- Project Human Resource Management
- Project Communication Management
- Project Risk Management
- Project Procurement Management
- Project Stakeholder Management

Finally, the 47 project management "processes" include topics such as (selected items):

- Develop Project Charter
- Develop Project Management Plan
- Direct and Manage Project Work
- Perform Integrated Change Control

Each process is categorized by one Process Group and one Knowledge Area, resulting in a matrix. A full matrix is not presented here due to copyright concerns, but one can be seen [here](#).

PMBOK details are easily obtained on the web, and will not be repeated here. It's clear that the Agile critiques of waterfall project management have been taken seriously by the PMBOK thought leaders. There is now a PMI Agile certification and much explicit recognition of the need for iterative and incremental approaches to project work.

PMBOK remains extensive and complex when considered as a whole. This is necessary, as it is used to manage extraordinarily complex and costly efforts in domains such as construction, military/aerospace, government, and others. Some of these efforts (especially those involving systems engineering, over and above software engineering) do have requirements for extensive planning and control that PMBOK meets well.

However, in Agile domains that seek to be more adaptive to changing business dynamics, full use of the PMBOK framework may be unnecessary and wasteful. The accepted response is to "tailor" the guidance, omitting those plans and deliverables that are not needed.

For example, PMBOK calls for a "stakeholder management plan" to be produced as part of the

Project Stakeholder Management knowledge area and the Planning process group. The author has rarely seen this deliverable formally produced, in years of enterprise experience. However, for certain (larger) projects it may make sense.

IMPORTANT

Part of the problem with extensive frameworks such as PMBOK is that knowing how and when to tailor them is hard-won knowledge that is not part of the usual formalized training. And yet, without some idea of "what matters" in applying the framework, there is great risk of wasted effort. The Agile movement in some ways is a reaction to the waste that can result from overly detailed frameworks.

The challenges of scope management

Scope management is a powerful tool and concept, at the heart of the most challenging debates around project management. PMBOK uses the following definitions [[PMI2013](#)]:

Scope. The sum of the products, services, and results to be provided as a project. See also project scope and product scope.

Scope Change. Any change to the project scope. A scope change almost always requires an adjustment to the project cost or schedule.

Scope Creep. The uncontrolled expansion to product or project scope without adjustments to time, cost, and resources.

Change Control A process whereby modifications to documents, deliverables, or baselines associated with the project are identified, documented, approved, or rejected.

In the [Lean Startup](#) world, products may pivot and pivot again, and their resource requirements may flex rapidly based on market opportunity. Formal project change control processes are in general not used. Even in larger organizations, product teams may be granted certain leeway to adapt their "products, services, and results" and while such adaptations need to be transparent, formal project change control is not the vehicle used.

On the other hand, remember our [emergence model](#). The simple organizational change from one to multiple products may provoke certain concerns and a new kind of contention for resources. People are inherently competitive and also have a sense of fairness. A new product team that seems to be unaccountable for results, consuming "more than its share" of budget while failing to meet the original vision for their existence, will cause conflict and concern among organizational leadership.

It is in the tension between product autonomy and accountability that we see project management techniques such as the work breakdown structure and project change control employed.

We saw a Work Breakdown Structure in the previous section, with our discussion of a holiday party. More formally, the work breakdown structure is defined by the Project Management Body of Knowledge as

...a hierarchical decomposition of the total scope of work to be carried out by the project team to accomplish the project objectives and create the required deliverables. The WBS organizes and defines the total scope of the project, and represents the work specified in the current approved project

[PMI2013].

[Portny2013] recommends "Subdivide your WBS component into additional deliverables if you think either of the following situations applies: The component will take much longer than two calendar weeks to complete. The component will require much more than 80 person-hours to complete."

This may seem reasonable, but in iterative product development, it can be difficult to "decompose" a problem in the way project management seems to require. Or to estimate in the way Portny suggests. This can lead to two problems.

First, the WBS may be created at a seemingly appropriate level of detail, but since it is created before key information is generated, it is inevitably wrong and needing ongoing correction. If the project management approach requires a high-effort "project change management" process, much waste may result as "approvals" are sought for each [feedback](#) cycle. This may result in increasing disregard by the development team for the project manager and his/her plan, and corresponding cultural risks of disengagement and lowering of trust on all sides.

Second, we may see the creation of project plans that are too high level, omitting information that is in fact known at the time - for example, external deadlines or resource constraints. This happens because the team develops a cultural attitude that is averse to all planning and estimation. (See the discussion on [No Estimates](#).)

Task ordering, dependencies, and critical path

The specific ordering of tasks is also important. As we saw in the holiday party example, sometimes one cannot complete a task without first completing something else - e.g., you cannot buy groceries until you decide on the menu. Or install software before the server is configured. But many other tasks might be performed in any order.

Sometimes, the tasks are constrained by the availability of someone to work on them. This is the situation we have in simple Kanban, where a backlog of tasks is worked primarily according to the availability of someone to work on them.

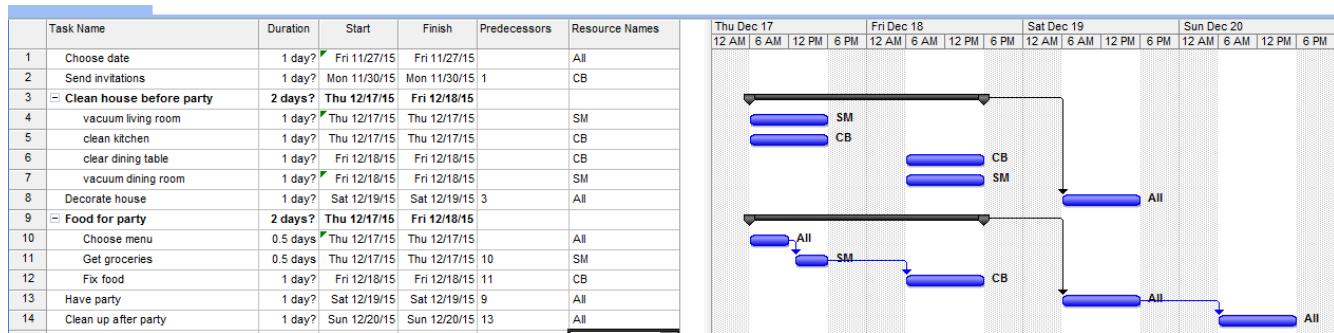
There are other approaches to scheduling tasks. [Portny2013] suggests the following approach types:

- Basic
- Procedural
- Legal/regulatory
- Optimization-based
- Arbitrary (no reason for precedence chosen, but once resources are planned, hard to change)

When tasks are ordered, and their time to complete estimated, an analysis known as **critical path** is possible.

PMBOK [PMI2013] defines the critical path as "The sequence of activities that represents the longest path through a project, which determines the shortest possible duration."

Consider our holiday party plan again:



In the above example, we have two major work areas: clean the house and prepare a menu. Note however that there are no dependencies between:

- vacuum living room
- clean kitchen
- clean dining table
- vacuum dining room

Really, they can be done in any order.

On the other hand, there is a clear set of dependencies between:

1. Choose menu
2. Get groceries
3. Fix food

This means that the sub-activity "Food for party" is the critical path. If the menu is not determined on time, the grocery run can't happen, and the food will be fixed late or not at all.

Loose coupling to the project plan rescue?

While this book does not go into systems architectural styles in depth, a project with a large number of dependencies may be an indication that the system or product being constructed also has significant interdependencies.

Successful systems designers for years have relied on concepts such as encapsulation, abstraction, and loose coupling to minimize the dependencies between components of complex systems, so that their design, construction, and operation can be managed with some degree of independence. These ideas are core to the software engineering literature.

Recent expressions of these core ideas are Services-Oriented Architecture and microservices.

Systems that do not adopt such approaches are often termed "monolithic" and have a well deserved reputation for being problematic to build and operate. Many large software failures stem from such approaches. (cites)

The system development efforts that gave rise to Critical Path Analysis (in particular the Polaris submarine) were very different from modern digital product development. A submarine is a relatively closed system with many constraints. They are changed only through the most formal of processes. (Even so, military systems seek loose coupling wherever possible.) Modern digital systems, while presenting their own challenges, do not operate under such constraints.

If you have a project plan with excessive dependencies, the question at least should be asked: does my massive, tightly-coupled project plan indicate I am building a monolithic, tightly-coupled system that will not be flexible or responsive to change?

Again, many digital companies build tremendously robust integrated services from the combination of many quasi-independent "product" teams, each serving a particular function.

The book [Small Pieces Loosely Joined: A Unified Theory Of The Web](#) is a good discussion of these ideas.

However, when a particular organizational objective requires changes to more than one such "product," the need for cross-team coordination emerges. Someone needs to own this larger objective, even if its actual implementation is carried out across multiple distinct teams.

In this discussion we see the emergence of enterprise architecture, which will be discussed in depth in Chapter 12.

Risk management

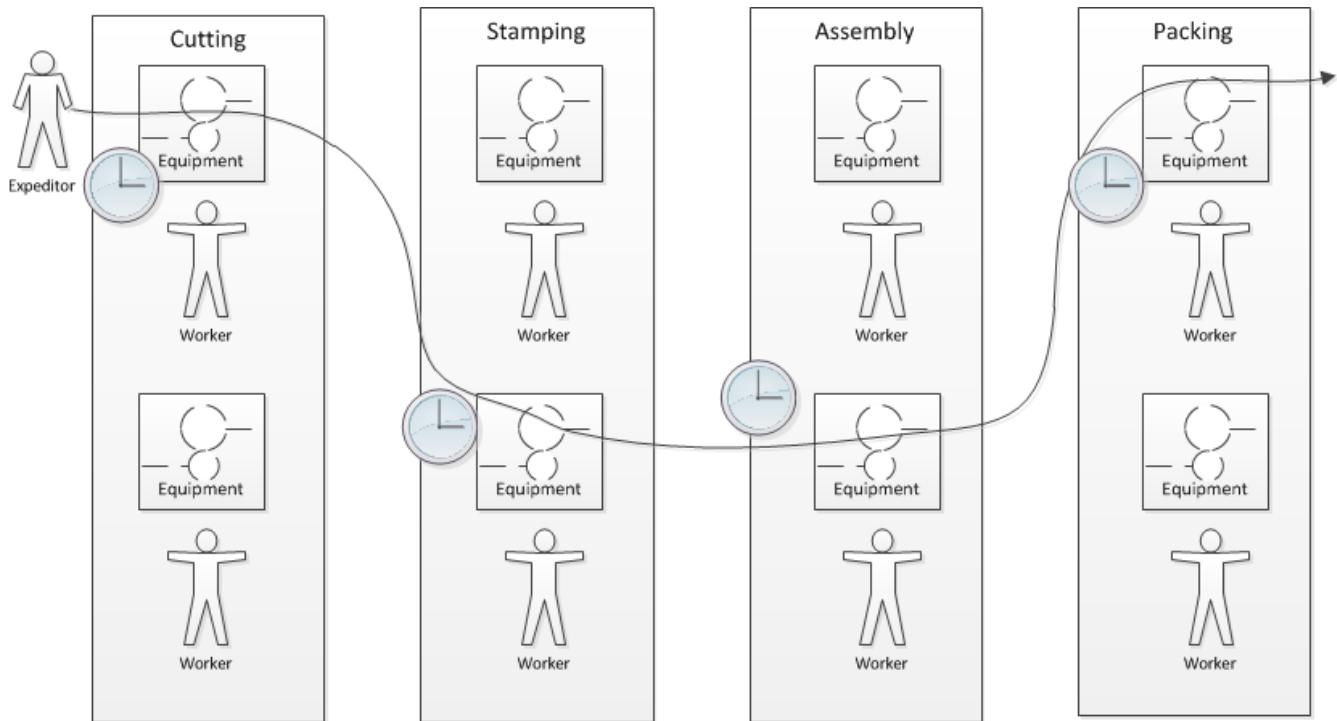
Project management is where we see the first formalization of risk management (which will be more extensively covered in Chapter 10). Briefly, risk is classically defined as the probability of an adverse event times its cost. Project managers are alert to risks to their timelines, resource estimates, and deliverables.

Risks may be formally identified in project management tooling. They may be accepted, avoided, transferred, or mitigated. Unmanaged risks to a project may result in the project as a whole reporting an unfavorable status.

Scheduling and execution

not sure about this section. Sets up the Critical Chain discussion. Also helpful to Chapter 9.

To understand how project management works across a functionally organized system, consider the internals of each functional area:



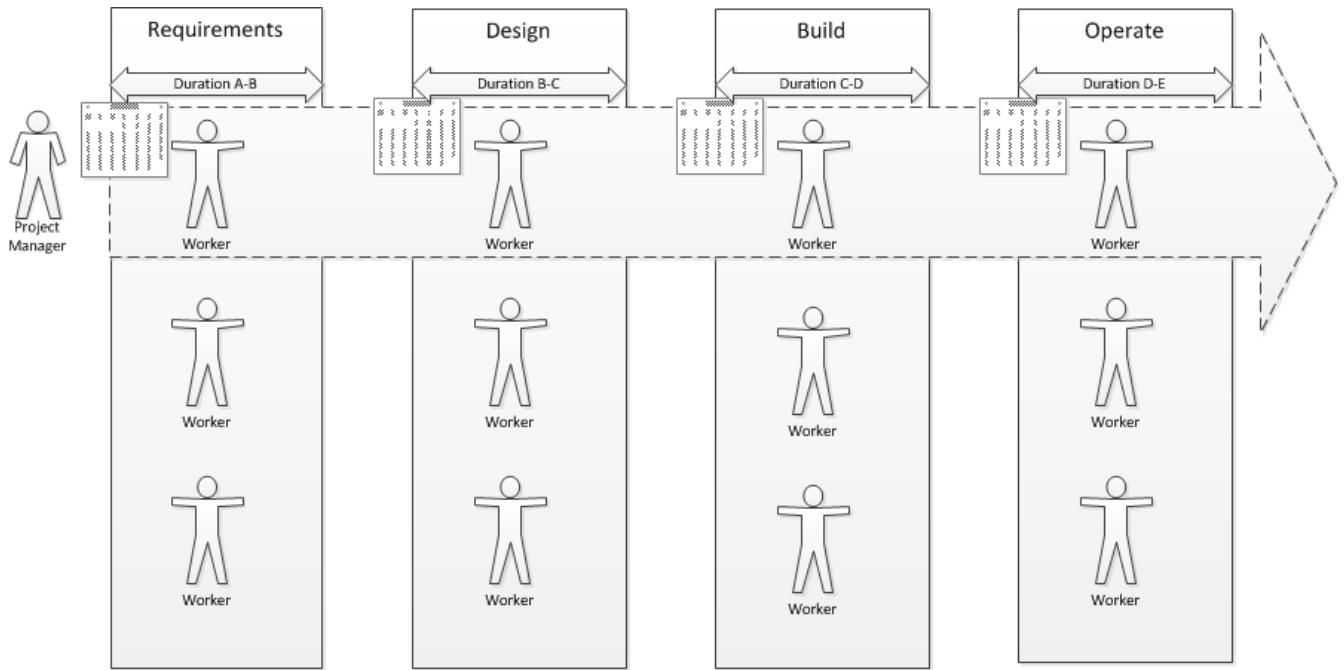
Each combination of worker and equipment can be considered a workstation in this simplified model.

If there is concern for flow across the functional areas, one technique used in manufacturing is that of expediting. An individual in the role of "expeditor" tracks the process across the work stations and ensures that any friction or blockages are overcome so that the entire process completes and value is delivered. This can be seen as a preliminary form of project management.

As the production line evolves, scheduling may be used as well: the assignment of work to each workstation (combination of worker and equipment) may have its timing and duration specified in advance. While this is not an optimal approach according to Lean philosophy (which emphasizes "pull" techniques), [production scheduling](#) is widely used in manufacturing.

NOTE Production scheduling is a large topic in and of itself. It is part of the domain of interest of the professional organization, The American Production and Inventory Control Society (APICS).

IT systems development, when crossing functions, is rarely if ever considered as a simple end to end process that can run itself, with only a lightweight expeditor. Instead, since there is substantially increased complexity, a project paradigm is used across the functions. The project manager plays the role of expanded expeditor:



The above diagram, which you should study carefully, is a more detailed representation of the classic "waterfall" project. It still bears strong similarities to the idealized manufacturing model presented just previously. In particular, notice that the default model is that **the need for the resource is tightly bounded by the calendar**. In the above model, the following resource requirements are apparent:

1. One Requirements analyst is needed between times A and B (e.g., from May 1 through 15).
2. One Architect is needed between times B and C (e.g., from May 16 through 31)
3. One Developer is needed between times C and D (e.g., from June 1 through 15)
4. One Release Engineer is needed between times D and E (e.g., from June 15 through 30)

This is a deliberately unworkable model for illustration. (But the author can attest it has been tried...) Sometimes, the terms "mechanistic," "deterministic," or "linear" are used to describe waterfall project methods. This shows them at their extreme, insofar as they are essentially translations of a manufacturing model.

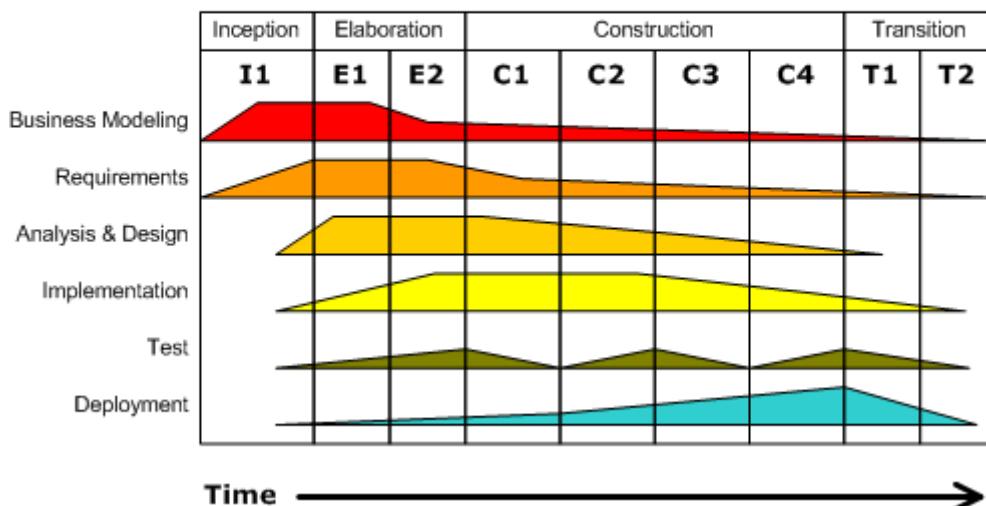
The basic problem: if the Developer is released from the project on June 15, and issues were to arise on June 20 with the release, flow would be interrupted and the system would not be delivered successfully.

Because of this, the project may retain some residual claim on all the engaged resources, but their utilization may be low or zero at certain points.

If we re-imagine the diagram with the different functional areas along the horizontal, we start to develop an understanding of the interrelatedness of product development. The following is one representation of the Rational Unified Process, an early depiction of this interrelatedness:

Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



Rational Unified Process, from [Wikipedia](#)

One interesting aspect of the diagram is that it shows just how difficult it can be to staff a project. Demand for given functional roles ebbs and flows unpredictably. The project manager is expected to manage the uncertain handoffs between the various functional areas, and anticipate resource requirements stemming from unexpected feedback demands. This is why "T-shaped" professionals (see Chapter 7) are desirable, because they can move from area to area in response to demand.

Ultimately, the reason classical production scheduling is inappropriate for delivering new IT functionality is that there is simply too much variability in [product development](#). (Again see Chapter 4 for further discussion of this.) However, this does not change the fact that some level of planning and estimation is needed. To what extent this more variable IT work can be quantitatively managed is an active area of discussion in the software, Agile, and related communities.

Effort vs. duration

Production spectrum illustration? (diagonal)

Projects and their resources

NOTE

The use of the term "resources" to refer to the people working on a project is often criticized as insensitive and inappropriate. However, it is standard terminology which this book therefore needs to reflect, as an introductory survey text. You are encouraged to NOT use this terminology as you progress in your career.

Modern enterprise IT organizations use a mix of project management, processes, and ad-hoc work routing to achieve their results. Often, resources (people) are assigned to multiple projects, a practice sometimes called "fractional allocation."

In fractional allocation, a database administrator will work 25% on one project, 25% on another,

and still be expected to work 50% on ongoing production support.

This may appear to work mathematically, but practically it is an ineffective practice. Both Gene Kim in The Phoenix Project [Kim2013] and Eli Goldratt in Critical Chain [Goldratt1997] present dramatized accounts of the overburden and gridlock that can result from such approaches.

Human beings are notably bad at multi-tasking, and the mental "context-switching" required to move from one task to another is wasteful and ultimately not scalable. A human being fractionally allocated to more and more projects will get less and less done in total, as the transactional friction of task switching increases.

More on this in the following section on resource management.

there have got to be good cites on this

Critical chain

Goldratt in the above-cited book Critical Chain develops a sophisticated critique of project estimation and the dysfunctions it promotes.

In a project requiring contributions from multiple skilled resources, a common practice is to ask each person, "how long will this take you?" The project manager then works the resulting estimates into the overall project plan.

The problem with this is that most people will estimate their time conservatively; they will forecast a longer duration than they actually require. When all these "padded" estimates are added together, the project may be unacceptably long. The agreed work will tend to expand to fill the time available (Parkinson's Law). Furthermore, most people will wait until the end of their window to perform their task - a person who asks for 3 weeks to perform one week of work will often not start until week 3. (Otherwise known as Student Syndrome.)

One of the reasons that people estimate conservatively is that project managers tend to be quite concerned if committed tasks are not performed on time. Failure to make the "deliverable" by the committed date may result in negative feedback to the employee's manager and resulting poor performance reviews.

Coupled with the above-cited drive to multi-tasking, these factors result in poor project performance, despite the array of modern project management techniques.

Goldratt suggested an alternate approach, in which the idea of "critical path" is enhanced with resource awareness. That is to say, the issue of timing and dependencies (itself a complex problem) is further enriched with the availability of resources to perform the work. (In general, the availability of assigned project resources is assumed, but this is not a wise assumption in project-centric environments.)

Estimation is handled more probabilistically, and the "critical chain" is **the combination of the critical path plus the resource assigned to complete the most critical task**. The theory is that a person performing such a task must be protected from distraction, and in fact project managers must expand their tools to effectively forecast and plan the critical chain.

This leads to some complex math, in particular a known problem called the Resource-Constrained Scheduling Problem. (e.g. http://www.iste.co.uk/data/doc_dtalmanhopmh.pdf) The fact that this problem is so notoriously difficult is indicative of the need for adaptive approaches; ultimately, rigorous analytic methods fail to cope with the complexity of such problems.

Craig Larman, in Scaling Lean and Agile Development, is sympathetic to the overall insights and goals of Critical Chain. However, with respect the full blown analytical approach it implies, he states

"We have seen two very large official "project management TOC" adoption attempts (and heard of one more) in companies developing software-intensive embedded systems... The practice was clearly heavy, not agile, and not lean. In all three cases, the approach was eventually found cumbersome and not very effective, and was dropped." [\[Larman2009\]](#)

Agile and project management

Projects of all types and sizes are now **the way** that organizations accomplish their work [emphasis added].

— Stanley Portny, Project Management for Dummies 4th ed.

...the project as a vehicle of IT execution has by and large, failed to live up to its promise of predictable delivery.

— Sriram Narayan, Scaling Agile: Problems and Solutions

Agile is having a profound impact on the project management profession and will cause us to fundamentally rethink many of the well-established notions of what a project manager is...

— Charles G. Cobb, The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach

As the above quotes indicate, there are diverse opinions on the role and importance of traditional project management in the enterprise. Clearly it is under pressure from the Agile movement. Project management professionals are advised to not deny or diminish this fact. One of the primary criticisms of project management as a paradigm is that it promotes large "batches" of work. It is possible for a modern, IT-centric organization to make considerable progress on the basis of **product management** plus simple **work management**, without the overhead of the formalized project lifecycle suggested by PMBOK.

Cloud is having impacts on traditional project management as well. As noted in the section on the **decline of traditional IT**, projects were often used to install vendor-delivered commodity software, such as for payroll or employee expense. Increasingly, that kind of functionality is delivered by online service providers, leaving "traditional" internal IT with considerably reduced responsibilities.

Some of the IT capability may remain in the guise of an internal "service broker," to assist with the

sourcing and procurement of online services. The remainder moves into digital product management, as the only need for internal IT is in the area of revenue-generating, market-facing strategic digital product.

So, this section will examine the following questions:

- Given the above trends, under what circumstances does formalized project management make economic sense?
- Assuming that formalized project management is employed, how does one continue to support objectives such as fast feedback and adaptability?

Scaling one product

Good team structure can go a long way toward reducing dependencies but will not eliminate them.

— Mike Cohn, *Succeeding with Agile*

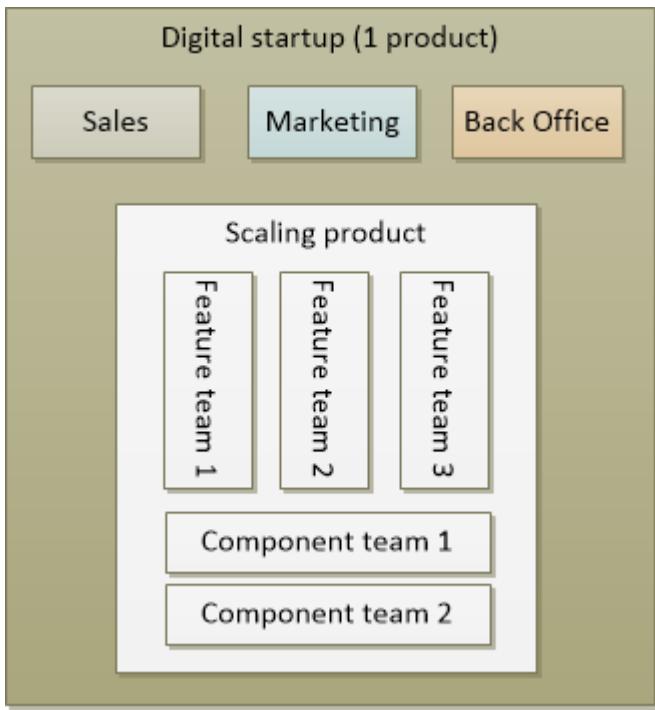
Where leveraging yellow stickies or index cards makes sense in conjunction with practices like big visible charts and co-location, such formats become ridiculous for a large constituency of challenging projects . . . When faced with these challenges, rather than proclaim that Agile won't work or doesn't scale, the preferable approach is to understand and acknowledge the nature of collaboration, the nature of distributed workflow, and the complexity of modern product development.

— Mark Kennaley, *SDLC 3.0*

What's typically underestimated is the complexity and indivisibility of many large-scale coordination tasks.

— Gary Hamel, preface to the *Open Organization: Igniting Passion and Performance*

As we discussed in the [Chapter 7](#), we are introducing multiple feature teams as our product scales up:



The decision process by which the features are identified is still informal (we will not formalize this until Chapter 12.) However, the term "architecture" is likely emerging through these discussions.

As noted in the discussion of [loose coupling and project management](#), some needs for coordination may be mitigated through the design of the product itself. If the features and components have well defined protocols for their interaction and clear contracts for matters like performance, development on each team can move forward with some autonomy.

The reality of microservices and loose coupling: the case of Chubby at Google

The idea that architecture can "emerge" across a set of smaller grained services ("microservices"), without explicit coordination, is attractive. Coordinating designs across products is expensive and ideally products should be able to evolve independently.

Mike Burrows of Google provides a detailed description of the Chubby lock service [[Burrows2006](#)], which is a prototypical example of a broadly-available internal service usable by a wide variety of other products.

The purpose of a lock service is to "allow its clients to synchronize their activities and to agree on basic information about their environment." Chubby was built from the start with objectives of reliability, availability to a "moderately large set of clients," and ease of understanding.

Burrows notes that even with such a cohesive and well-designed internal service, they still encounter coordination problems requiring human intervention. Such problems include:

- Use ("abuse") in unintended ways by clients
- Invalid assumptions by clients regarding Chubby's availability

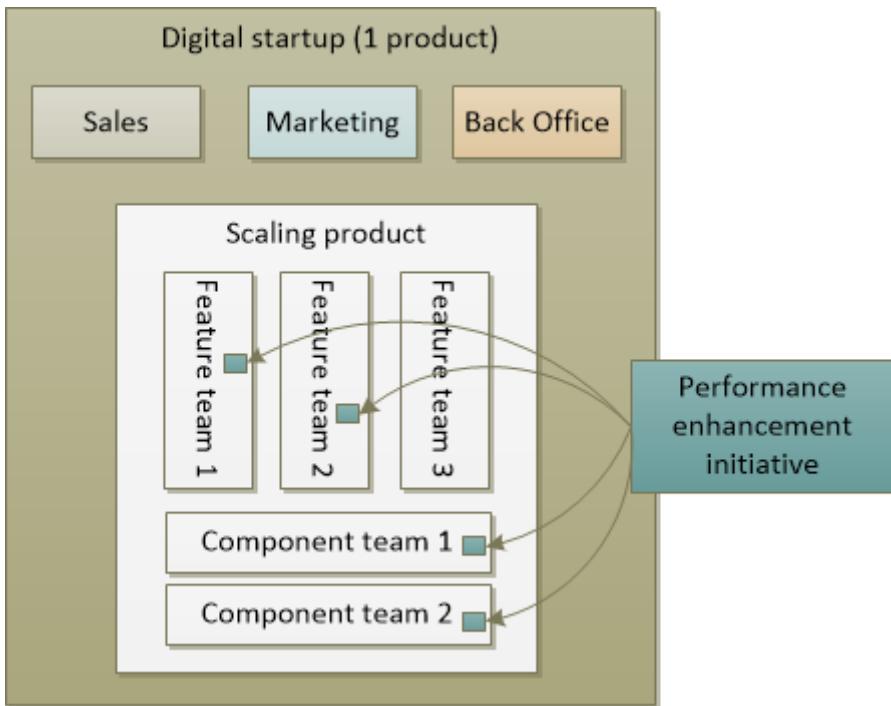
Because of this, the Chubby team (at least at the time writing of the case study) instituted a review process when new clients wished to start using the lock manager.

In terms of this chapter's topic, this means that someone on the product team needed to coordinate the discussions with the Chubby team and ensure that any concerns were resolved. This might conceivably have involved multiple iterations and reviews of designs describing intended use.

In short, even the most sophisticated microservice environments may require some level of human coordination across the teams. Project management provides a framework for doing this and encourages the development of appropriate skills.

But at scale, complexity is inevitable. What happens when a given business objective requires coordinated effort across multiple teams?

For example, an online ecommerce site might find itself overwhelmed by business success. Upgrading the site to accommodate the new demand might require distinct development work to be performed by most or all of the teams:



As the quote from Gary Hamel above indicates, a central point of coordination and accountability is advisable, otherwise the objective is at risk. (It becomes "someone else's problem.")

Scaling to more than one product

Directors should monitor the progress of approved IT proposals to ensure that they are achieving objectives in required timeframes using allocated resources.

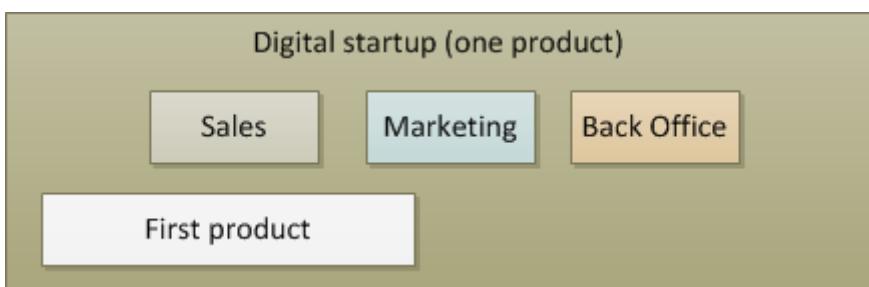
— ISO 38500:2008

disclaimer: the "fairness" driver is hypothetical

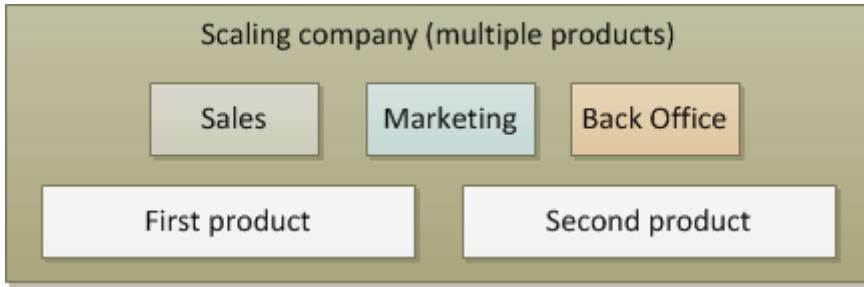
In keeping with the [emergence model](#) thought experiment, another key state transition is from one product to multiple products.

This is substantially distinct from scaling one product through subdividing it into multiple features.

Consider our hypothetical startup company:



Establishment of a second product team raises issues:



When everyone on the team is supporting one product and dedicated to its success, there is less sense of contention with "others" in the organization. This changes dramatically with the addition of a second product. Concerns for fair allocation and a sense of internal competition naturally arise out of this diversification.

Because resources are always limited, it is critical that the demands of each product be managed using objective criteria, requiring formalization. This was a different problem when you were a tight-knit startup; you were constrained, but everyone knew they were "in it together." Now you need some ground rules to support your increasingly diverse activities. This leads to new concerns:

- Managing scope and preventing unintended creep or drift from the product's original charter
- Managing contention for enterprise resources
- Execution to timeframes (e.g. the critical trade show)
- Coordinating dependencies (e.g. achieving larger, cross-product goals)
- Accountability for results

Fairness is deeply wired into the human brain [cites], and the creation of a new product (and associated team) provokes new dynamics in the growing company.

Why project management?

Project management provides tools addressing these issues, which is why it is widely used.

However, there are a number of issues in doing so.

- Projects are by definition temporary, while products may last as long as there is market demand.
- Project management methodology, with its emphasis on predictability, scope management, and change control often conflicts with the product management objective of discovering information (see the discussion of [Lean Product Development](#).)
- But not all product management activities involve the creation of new information!

Scope, execution, limited resources, and dependencies exist throughout digital business. A product manager with no understanding of these issues, or tools to deal with them, will likely fail.

Product managers should therefore be familiar with the basic concepts of project management. The project paradigm has a benefit in its explicit limitation of time and money, and the sense of urgency this creates. However, the way in which project management is implemented, the degree of formality, will vary according to need.

So, what of project management in this new world? A project manager may still be required, to facilitate discussions, record decisions, and keep the team on track to its stated direction and commitments. Regardless of whether the team considers itself "Agile," people are sometimes bad at taking notes or being consistent in their usage of tools such as Kanban boards and standups.

It is also useful to have a third party who is knowledgeable about the product, yet has some emotional distance from its success. This can be a difficult balance to strike, but the existence of the role of Scrum coach is indicative of its importance.

Governing outsourced work

A third major reason for the continued use of project management and its techniques is governing work that has been outsourced to third parties. This is covered in detail in the section on [sourcing](#).

Organizational tools and techniques

Our previous discussion of [work management](#) was a simple, idealized flow of uniform demand (new product functionality, issues, etc). Tasks in general did not have dependencies, or dependencies were handled through ad-hoc coordination. We also in general assumed that resources (people) were available to perform the tasks; resource contention, while it certainly may have come up, was again handled through ad-hoc means.

As we progress on our journey and continue to scale up, we see that dependencies and resource management have become defining concerns, and the concept of "project management" is a primary response. However, we retain our concern for fast feedback and adaptability, as well as a critical approach to the idea that complex initiatives can be precisely defined and simply executed through [open loop](#) approaches.

Above, we've discussed a few aspects of the coordination problem including the emergent issues when needed capabilities span feature teams, and when new product lines are introduced. In this section, we will discuss some of the organizational responses (techniques and tools) that have emerged as proven responses to these emergent issues.

The general problem is, "how do I facilitate the adaptive generation of information under conditions of time and space shifting." That is to say:

- digital product development (as all R&D) is essentially [information generation](#)
- as it scales up, we encounter emergent coordination needs
- that present rapidly increasing combinatoric complexity
- that may be exacerbated by time and space shifting

We have seen earlier versions of this problem in our discussions of work management. However, simple Kanban and visual Andon is no longer sufficient, given the nature of the coordination we now require. We need a more diverse and comprehensive set of techniques.

IMPORTANT

The discussion of particular techniques is always hazardous. People will tend to latch on to a promising approach without fully discussing the "target operating condition." Review the discussion of [Toyota Kata](#). As noted by [\[Larman2009\]](#) (p.44), the risk is one of [cargo cult](#) process adoption.

The primary objective remains one of "coordination" (as suggested by the overall name of this section, [Coordination](#).)

Here are a few techniques suggested by various Agile authors (primarily [\[Cohn2010\]](#))

Co-location

Don Reinertsen ([\[Reinertsen2009\]](#), p. 230), proposes "The Principle of Colocation" which asserts that "Colocation improves almost all aspects of communication." In order to scale this, one logically needs what Mike Cohn ([\[Cohn2010\]](#), p. 346) calls "The Big Room."

In terms of communications, this has significant organizational advantages. Communications are as simple as walking over to another person's desk, or just shouting out over the room. It is also easy to synchronize the entire room, through calling for everyone's attention.

However, there are limits to scaling the "Big Room" approach:

- Contention for key individual's attention
- "All hands" calls for attention that actually interest only a subset of the room
- Increasing ambient noise in the room
- Distracting individuals from intellectually demanding work requiring concentration, and ultimately interfering with their [personal sense of flow](#) - a destructive outcome.

The tension between team coordination and individual focus will likely continue. It is an ongoing topic in facilities design.

[recent popular press about problems of open space working, and responsive facilities with various workspace types available on reservation]

Synchronization points

If the team cannot work all the time in one room, then perhaps they can at least be gathered periodically. There is a broad spectrum of synchronization approaches:

- Daily standups
- Weekly status meetings
- Coordination meetings (e.g. Scrum of Scrums, see below)
- Release kickoffs
- Quarterly "all-hands" meetings
- Cross-organizational advisory and review boards

- Open Space inspired "unmeetings" and "unconferences"

All of them are essentially similar in approach and assumption: build a shared understanding of the work, objectives, or mission among smaller or larger sections of the organization, through limited-time face to face interaction, often on a defined time interval.

Meetings bloody meetings

Sidebar: Practical collaboration at scale (Chapter 7??)
Unmeetings/unconferences/Open space

reconsider relationship w/next section - isn't this a specific instance of a coordination structure

Liaison and coordination structures

The philosophy is that you push the power of decision making out to the periphery and away from the center. You give people the room to adopt, based on their experiences and expertise. All you ask is that they talk to one another and take responsibility. That is what works.

— Atul Gawande, The Checklist Manifesto

A variety of liaison and coordination approaches are suggested by Mike Cohn ([\[Cohn2010\]](#), Chapter 17, "Scaling Scrum") including:

- Shared team members
- Integration teams
- Communities of practice
- Scrum of scrums

Refactor this. Include submittal schedules in list. Find out Cohn's original sources. Other techniques?

Shared team members are suggested when two teams have a persistent interface requiring focus and ownership. When a product has multiple interfaces that emerge as a problem requiring focus, an **integration team** may be called for. **Communities of practice** were introduced in Chapter 7 as we discussed the [Spotify model](#). Considered here, they may also play a coordination role as well as a practice development/maturity role.

Finally, the idea of a "scrum of scrums" is essentially a representative or delegated model, in which

each individual Scrum team sends one individual to a periodic coordination meeting where matters of cross-team concern can be discussed and decisions taken.

[Cohn2010] (p342) cautions: "A scrum of scrums meeting will feel nothing like a daily scrum despite the similarities in names. The daily scrum is a synchronization meeting: individual team members come together to communicate about their work and synchronize their efforts. The scrum of scrums, on the other hand, is a problem-solving meeting and will not have the same quick, get-in-get-out tone of a daily scrum."

Another perspective on this is found in The Checklist Manifesto [Gawande2010]. Some work, while detailed, can be planned to a high degree of detail (i.e. the "checklists" of the title). However, emergent complexity requires different approach - no checklist can anticipate all eventualities. To handle all the emergent complexity, the coordination focus must shift to structuring the right communications. In examining modern construction industry techniques, Gawande noted the concept of the "submittal schedule," which "didn't specify construction tasks; it specified *communication* tasks" (p. 65, emphasis supplied).

Planning techniques

Techniques of co-location, synchronization, and liaison all support to varying degrees a concept of planning. Fundamentally, we plan so that the work progresses effectively and efficiently. We seek to optimize the following contradictory goals:

- Delivering results quickly
- Minimizing the waste of un-utilized resources

To meet these goals, we

- estimate so that we have some understanding of our resource needs
- plan so that we understand dependencies (e.g. when one team **must** complete a task before another team can start theirs)

The above discussion is by no means exhaustive. A wealth of additional techniques is to be found in [Larman2009] and [Cohn2010]. New techniques are continually emerging from the front lines of the digital industry; the interested student should consider attending industry conferences such as those offered by the Agile Alliance.

The end of the "iron triangle?"

cite Edwards & others - focus on throughput moves the needle on other metrics - but not the reverse.
Humble toyota quote

The future of project management

Recall our three "Ps":

- Product

- Project
- Process

Taken together, the three represent a coherent set of concerns for value delivery in various forms. But in isolation, any one of them ultimately is limited. This is a particular challenge for project management, whose practitioners may identify deeply with their chosen field of expertise.

Clearly, formalized project management is under pressure. Its methods are perceived by the Agile community as overly heavyweight; its practitioners are criticized for focusing too much on success in terms of cost and schedule performance and not enough on business outcomes. Because projects are by definition temporary, project managers have little incentive to care about technical debt or operational consequences. Hence the rise of the product manager.

However, a product manager who does not understand the fundamentals of project execution will not succeed. As we have seen, modern products, especially in organizations scaling up, have dependencies and coordination needs, and to meet those needs, project management tools will continue to provide value.

2nd draft incorporate <http://blackswanfarming.com/fund-the-teams-not-the-project/>

IT sourcing and contract management

IT sourcing is the set of concerns related to identifying suppliers (sources) of the necessary inputs to deliver digital value. **Contract management** is a critical, subsidiary concern, where digital value intersects with law.

The basic classes of inputs include:

- People (with certain skills and knowledge)
- Hardware
- Software
- Services (which themselves are composed of people, hardware, and/or software)

Practically speaking, these inputs are handled by two different functions:

- People (in particular, full time employees) are handled by a Human Resources function.
- Hardware, software, and services are handled by a Procurement function. Other terms associated with this are Vendor Management, Contract Management, and Supplier Management. We will not attempt to clarify or rationalize these areas in this section.

We will defer the question of people and their skills and knowledge to the next section.

Basic concerns

Case study: Choosing a telecommunications provider

When Company X was a startup, its telecommunications needs were limited, as were its options. The founder had one choice for Internet access, the local cable company. Even when the company moved to a larger space, as a single team startup, its options were limited.

However, it is now a company of 50, and moving yet again to a new headquarters where there are a variety of options for network carriers. The company is known to be growing and three telecommunications companies ("carriers") have been sending sales representatives periodically to inquire if their services might be needed.

With the move to a new facility, some systematic effort must be undertaken to choose an appropriate provider. This becomes a sub-project in its own right, part of the larger program required to effectively complete the move.

As part of this project, a formal "Request for Information" (RFI) is sent to all the potential carriers. Part of this RFI consists of a lengthy series of questions, such as:

- What kinds of circuits are available?
- What is their territory?
- How much data can they handle?
- What do they cost (at a high level)?
- How are they secured?
- How stable are they (how often are they "down")?
- Are co-location services available (can the carrier host the company's servers in its data centers?)
- What other services does the carrier provide?

The responses to these questions are "scored" (assigned a numeric weighting) and the 2 top scoring vendors are issued a Request for Quote (RFQ). The RFQ goes into much more detail in terms of the contract the carrier is willing to offer. After extensive discussions and negotiations, Company X's contract team awards the business to the carrier they believe will provide the greatest value.

The same approach is used to establish relationships with Cloud vendors, software providers, and consultants. In fact, because the approach is so consistent, it is considered a repeatable "process." See the chapter on [process management](#).

A small company may establish binding agreements with vendors relatively casually. For example, when the founder first chose a Cloud platform on which to build their product, they clicked on a button that said "I accept," at the bottom of a lengthy legal document they didn't necessarily read closely. This "clickwrap" agreement is a legally binding contract, which means that the terms and conditions it specifies are enforceable by the courts.

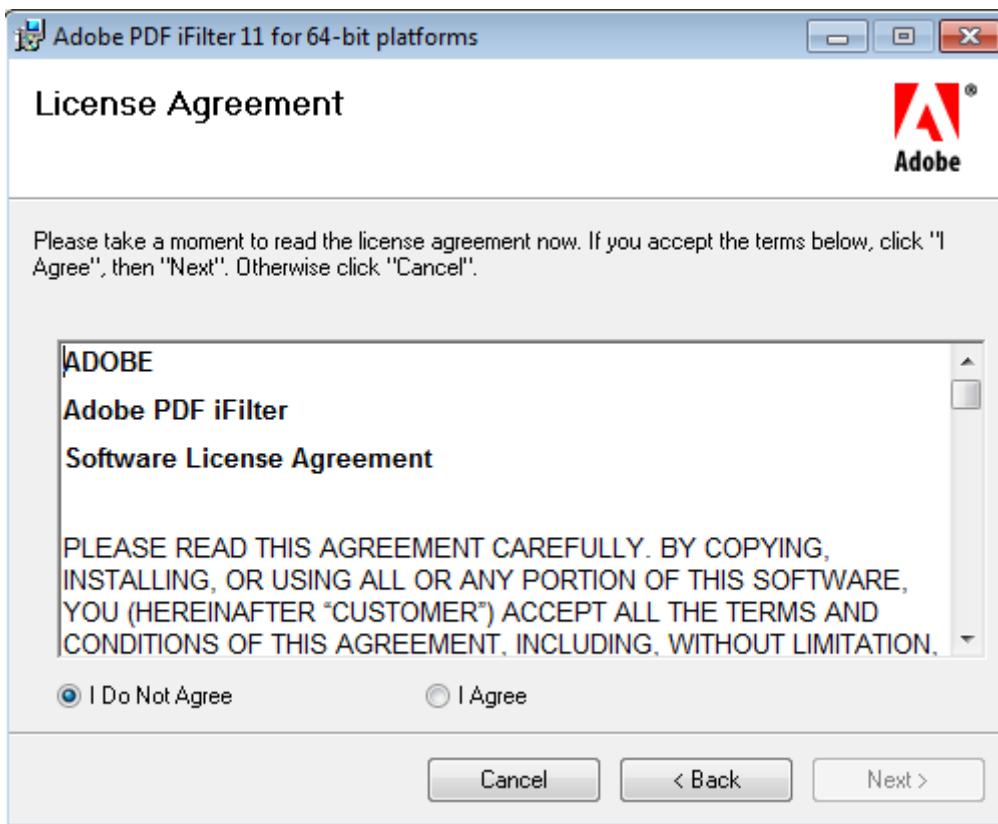


Figure: Example of clickwrap licensing

A startup may be inattentive to the full implications of its contracts for various reasons:

- The founder does not understand the importance and consequences of legally binding agreements
- The founder understands, but feels they have little to lose (for example, they have incorporated as a limited liability company, meaning the founder's personal assets are not at risk)
- The service is perceived to be so broadly used that an agreement with it must be safe (if 50 other startups are using a well known Cloud provider and prospering, why would a startup founder spend precious time and money on overly detailed scrutiny of its agreements?)

All of these assumptions bear some risk—and many startups have been burned on such matters—but there are many other, perhaps more pressing risks for the founder and startup. See the case study on the [9 figure true-up](#) for a grim scenario.

However, by the time the company has scaled to the team of teams level, contract management is almost certainly a concern of the Chief Financial Officer. The company has more resources ("deeper pockets") and so lawsuits start to become a concern. The complexity of the company's services may require more specialized terms and conditions. Standard "boilerplate" contracts thus are replaced by individualized agreements. Concerns over intellectual property, the ability to terminate agreements, liability for damages, and other topics require increased negotiation and counterproposing contractual language.

At this point, the company may have hired its own legal professional; certainly, legal fees are going up, whether as services from external law firms or internal staff.

Contract and vendor management is more than just establishing the initial agreement. The ongoing

relationship must be monitored for its consistency with the contract. If the vendor agrees that its service will be available 99.999% of the time, the availability of that service should be measured and if it falls below that threshold, contractual penalties may need to be applied.

In larger organizations, where a vendor might hold multiple contracts, a concept of "Vendor Management" emerges. Vendors may be provided "scorecards" that aggregate metrics describing their performance and the overall impression of the customer. Perhaps key stakeholders are internally surveyed as to their impression of the vendor's performance and whether they would be likely to recommend them again. Low scores may penalize a vendor's chances in future RFI/RFP processes; high scores may enhance them. Finally, advising on sourcing is one of the main services an [Enterprise Architecture](#) group may provide.

Outsourcing and Cloudsourcing

cloud computing has some fundamental characteristics that distinguish it from traditional outsourcing...many cloud services are merely passive providers of computing resources, utilized by users to perform their own processing.

— Millard et al, Cloud Computing Law

Software development is only one form of contracted service. Another major class of contract in the modern digital organization is for [Cloud](#) services.

- lock-in
- insurance
- <http://jpmorgenthal.com/2016/08/24/a-reality-check-on-everyones-moving-everything-to-the-cloud/>

Software development and contracts

Customer collaboration over contract negotiation.

— Agile Manifesto

For both suppliers and buyers of information technology (IT) projects, one issue arises repeatedly: how to get out of the trap of fixed pricing without the disadvantages of time and materials contracts.

— Andreas Opelt et al, Agile Contracts: Creating and Managing Successful Projects with Scrum

What do lawyers assume is the nature of software projects? First, it is common that they view it as similar to a construction project—relatively predictable—rather than the highly uncertain and variable research and development that it usually is. Second, that in the project (1) there is a long delay before something can be delivered that is well done, with (2) late and weak feedback, (3) long payment cycles, and (4) great problems for the customer if the project is stopped at any arbitrary point in time. **These assumptions are invalidated in agile development.**

— Arbogast et al, Agile Contracts Primer

Software is often developed and delivered per contractual terms. Contracts are legally binding agreements, typically developed with the assistance of lawyers. As noted in [Arbogast2012] (p.5), "Legal professionals are trained to act, under legal duty, to advance their client's interests and protect them against all pitfalls, seen or unseen." The idea of "customer collaboration over contract negotiation" may strike them as the height of naivete. However, Agile and Lean influences have made substantial inroads in contracting approaches.

Arbogast et al. describe the general areas of contract concern:

- Risk, exposure, and liability
- Flexibility
- Clarity of obligations, expectations, and deliverables

They argue that "An agile-project contract may articulate the same limitations of liability (and related terms) as a traditional-project contract, but the agile contract will better support avoiding the very problems that a lawyer is worried about." (p. 12)

So, what is an agile contract?

There are two major classes of contracts:

- Time and materials
- Fixed price

In a **time and materials** contract, the contracting firm simply pays the provider until the work is done. This means that the risk of the project overrunning its schedule or budget resides primarily with the firm hiring out the work. While this can work, there is often a desire on the part of the firm to reduce this risk. If you are hiring someone because they claim they are experts, and can do the work better/cheaper/quicker than your own staff, it seems reasonable that they should be willing to shoulder some of the risk.

In a **fixed price** contract, the vendor providing the service will make a binding commitment that (for example) "we will get the software completely written in 9 months for \$3 million." Penalties may be enforced if the software is late, and it's up to the vendor to control development costs. If the vendor does not understand the work correctly, they may lose money on the deal.

Reconciling Agile with fixed-price contracting approaches has been a challenging topic [Opelt2013]. The desire for control over a contractual relationship is historically one of the major drivers of [waterfall](#) approaches. However, since requirements cannot be known in advance, this is problematic.

When a contract is signed based on waterfall assumptions, the project management process of [change control](#) is typically used to govern any alterations to the scope of the effort. Each change order typically implies some increase in cost to the customer. Because of this, the perceived risk mitigation of a fixed price contract may become a false premise.

This problem has been understood for some time. Scott Ambler argued in 2005 that "It's time to abandon the idea that fixed bids reduce risk. Clients have far more control over a project with a variable, gated approach to funding in which working software is delivered on a regular basis" [Ambler2005]. Andreas Opelt states, "For agile IT projects it is therefore necessary to find an agreement that supports the balance between a fixed budget (maximum price range) and agile development (scope not yet defined in detail)..."

How is this done? Opelt and his co-authors further argue that the essential question revolves around the project "iron triangle":

- Scope
- Cost
- Deadline

The approach they recommend is determining which of these elements is the "fixed point" and which is estimated. In traditional waterfall projects, scope is fixed, while costs and deadline must be estimated (a problematic approach when [product development](#) is required.)

In Opelt's view, in Agile contracting, costs and deadline are fixed, while scope is "estimated" - understood to have some inevitable variability. "...you never know exactly what details will be needed at the start of a project. On the other hand, you do not always need everything that had originally been considered to be important" [Opelt2013].

Their recommended approach supports the following benefits:

- Simplified adaptation to change
- Non-punitive changes in scope
- Reduced knowledge decay (large "batches" of requirements degrade in value over time)

This is achieved through:

- Defining the contract at the level of product or project vision (epics or high-level stories; see discussion of [Scrum](#)) - not detailed specification
- Developing high-level estimation
- Establishing agreement for sharing the risk of product development variability

This last point, which Opelt et al term "riskshare," is key. If the schedule or cost expand beyond the initial estimate, both the supplier and the customer pay, according to some agreed %, which they

recommend be between 30%-70%. If the supplier proportion is too low, the contract becomes essentially time & materials. If the customer proportion is too low, the contract starts to resemble traditional fixed-price.

Incremental checkpoints are also essential; for example, the supplier/customer interactions should be high bandwidth for the first few sprints, while culture and expectations are being established and the project is developing a rhythm.

Finally, the ability for either party to exit gracefully and with minimum penalty is needed. If the initiative is testing market response (ala [Lean Startup](#)) and the product hypothesis is falsified, there is little point to continuing the work from the customer's point of view. AND, if the product vision turns out to be far more work than either party estimated, the supplier should be able to walk away (or at least insist on comprehensive re-negotiation.)

These ideas are a departure from traditional contract management. As Opelt asks, "How can you sign a contract from which one party can exit at any time?" Recall however that (if Agile principles are applied) the customer is receiving working software continuously through the engagement (e.g. after every sprint).

In conclusion, as Arbogast et al argue, "Contracts that promote or mandate sequential life cycle development increase project risk... an agile approach ... reduces risk because it limits both the scope of the deliverable and extent of the payment [and] allows for inevitable change" [\[Arbogast2012\]](#) (p.13).

Agile 2016 concepts
"iterative contracts"
"fixed profit contract"
"don't want to share margins" problem - offer sprints @ discount

Software licensing

As software and digital services are increasingly used by firms large and small, the contractual rights of usage become more and more critical.

We mentioned a "clickwrap" licensing agreement above. Software licensing in general is a large and detailed topic, one presenting substantial financial risk to the unwary firm, especially when Cloud and virtualization are concerned.

Case study: The 9-figure "true-up" A large enterprise had a long relationship with a major software vendor, who provided a critical software product used widely for many purposes across the enterprise.

The price for this product was set based on the power of the computer running it. A license would cost less for computer with 4 cores and 1 gigabyte of RAM, than it would for a computer with 16 cores and 8 gigabytes of RAM. The largest computers required the most expensive licenses.

As described previously, the goal of [virtualization](#) is to use one powerful physical computer to consolidate more lightly-loaded computers as "virtual machines." This can provide significant savings.

Over the course of 3 years, the enterprise described here virtualized about 5,000 formerly physical computers, each of which had been running the vendor's software.

However, a deadly wrinkle emerged in the software vendor's licensing terms. The formerly physical computers were in general smaller machines. The new virtual farms were clusters of 16 of the most powerful computers available on the market. The vendor held that EACH of the 5,000 instances of its software running in the virtual machines was liable for the FULL licensing fee applicable to the most powerful machine!

Even though each of the 5,000 virtual machines could not possibly have been using the full capacity of the virtual farm, the vendor insisted (and was upheld) that the contract did not account for that, and there was no way of knowing whether any given VM had been using the full capacity of the farm at some point.

The dispute escalated to the CEOs of each company, but the vendor held firm. The enterprise was obliged to pay a "true-up" charge of over \$100 million (9 figures).

This is not an isolated instance. Major software vendors have earned billions in such charges and continue to audit aggressively for these kinds of scenarios. This is why contracts and licenses should never be taken lightly. Even startups could be vulnerable, if licensed commercial software is used in un-authorized ways in a Cloud environment, for example.

Steve Russman quote

Software licensing is a subset of Software Asset Management, which is itself a subset of IT Asset Management, discussed in more depth in the material on [process management](#) and [IT lifecycles](#).

FOSS licensing... risks etc ... solicit contribution

Bente mentions Linux and Hibernate issues that almost killed them - was Linux part of SCO thing?

Cross-reference to risk mgmt section, also bring in package mgmt. You have been doing it, right???

FOSS as IT performance factor, see Forsgren/Humble

<https://www.youtube.com/watch?v=cJVUtBSmXaM>

The role of industry analysts

When a company is faced by a sourcing question of any kind, one initial reaction is to research the market alternatives. But research is time consuming and markets are large and complex. Therefore, the role of industry or market analyst has developed.

In the financial media, one often hears from "industry analysts" who are concerned with whether a company is a good investment in the stock market. While there is some overlap, the industry analysts we are concerned with here are more focused on advising prospective customers of a given market's offerings.

Because sourcing and contracting is an infrequent activity, especially for smaller companies, it is valuable to have such services. Because they are researching a market and talking to vendors and customers on a regular basis, analysts can be helpful to companies in the purchasing process.

However, analysts take money from the vendors they are covering as well, leading to occasional charges of conflict of interest [cite]. How does this work? There are a couple of ways.

First, the analyst firm engages in objective research of a given market segment. They do this by developing a common set of criteria for who gets included, and a detailed set of questions to assess their capabilities.

For example, an analyst firm might define a market segment of "Cloud Infrastructure as a Service" vendors. Only vendors supporting the basic NIST guidelines for Infrastructure as a Service are invited. Then, the analyst might ask, "Do you support Software Defined Networking, e.g. Network Function Virtualization" as a question. Companies that answer "yes" will be given a higher score than companies that answer "no." The number of questions on a major research report might be as high as 300 or even higher.

Once the report is completed and the vendors are ranked (analyst firms typically use a two-dimensional ranking, such as the Gartner Magic Quadrant), it is made available to end users for a fee. Fees for such research might range from \$500 to \$5000 or more, depending on how specialized the topic, how difficult the research, and the ability to pay of prospective customers.

get permission to use an MQ or similar

NOTE

Large companies, e.g. those in the Fortune 500, typically would purchase an "enterprise agreement", often defined as a named "seat" for an individual, who can then access entire categories of research.

Customers may have further questions for the analyst who wrote the research. They may be entitled to some portion of analyst time as part of their license, or they may pay extra for this privilege.

Beyond selling the research to potential customers of a market, the analyst firm has a complex relationship with the vendors they are covering. In our example of a major market research report, the analyst firm's sales team also reaches out the vendors who were covered. The conversation goes something like this:

"Greetings. You did very well in our recent research report. Would you like to be able to give it away to prospective customers, with your success highlighted? If so, you can sponsor the report for \$50,000."

Because the analyst report is seen as having some independence, it can be an attractive marketing tool for the vendor, who will often pay (after some negotiating) for the sponsorship. In fact, vendors have so many opportunities along these lines they often find it necessary to establish a function known as "Analyst Relations" to manage all of them.

===== Digital sourcing and SIAM

SIAM: Supplier integration and management

ask Finister

===== Reframing biz v it -> demand v supply

this is an important point that should be at the top

IT human resource management

Collaborative

Status: This section is 1st draft as of 2016-05-04.

Needs: As you review this, please keep in mind that it is a secondary section to the overall chapter "Project and Resource Management." Another case study would be OK and I think that the end doesn't quite take it home yet. Of course anything I have completely missed would be great feedback!

Resource management is a terrible name. We will use it anyways.

Todo: get permission for Dilbert cartoon "We're human beings not resources"
<http://dilbert.com/strip/1995-09-22>

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Basic concerns

Human Resource management can also be termed "people management." It is distinct from supply chain and technology management, being concerned with the identification and recruitment, onboarding, ongoing management for value, and eventual exit of individuals from an organization.

This brief section covers the topic as it relates to digital management, incorporating recent cases and perspectives.

Hiring

hiring is one of the most important things a software organization does. Every good hire accelerates your organization; every poor hire is a drag on your organization.

— Sean Landis, Agile Hiring

Here is a typical hiring process:

- Solicit candidates, through various channels such as job boards and recruiters

- Review resumes and narrow candidate pool down for phone interviews
- Conduct phone interviews and narrow candidate pool down for in person interviews
- Conduct in person interviews, identify candidates for offers
- Make offer, negotiate to acceptance
- Hire and onboard

Your organization has been hiring people for some time now. It's always been one of your most important decisions, but you have reached the point where a more formal and explicit understanding of how you do this is essential.

First, why do you hire new staff? How and when do you perceive a need? It is well established that increasing the size of a team can slow it down. Legendary software engineer Fred Brooks, in his work *The Mythical Man Month*, identified the pattern that "adding more people to a late project makes it later" [\[Brooks1995\]](#).

Are you adding people because of a perceived need for specialist skills? While this will always be a reason for new hires, many argue in favor of "T-shaped" people - people who are deep in one skill, and broad in others. Hiring new staff has an impact on culture - is it better to train from within, or source externally?

Second, how are you hiring staff? In a traditional, functionally specialized model, someone in a Human Resources organization acts as an intermediary between the hiring manager, and the job applicant (sometimes with a recruiter also in the mix between the company and the applicant). Detailed job descriptions are developed, and applicants not explicitly matching the selection criteria (e.g. in their resume) are not invited for an interview.

Such practices do not necessarily result in good outcomes. Applicants routinely tailor resumes to the job description. In some cases they have been known to copy the job description into invisible sections of their resume, so that they are guaranteed a "match" if automated resume-scanning software is used.

A compelling case study of the limitations of traditional HR-driven hiring is discussed by Robert Sutton and Huggy Rao in *Scaling up Excellence: Getting to More without Settling for Less*[\[Sutton2014\]](#). The authors describe the company Lotus Software, one of the pioneers of early desktop computing.

With [company founder] Kapor's permission, [head of organizational development] Klein pulled together the résumés of the first forty Lotus employees...[and] submitted all forty résumés to the Lotus human resources department. Not one of the forty applicants, including Kapor, was invited for a job interview. The founders had built a world that rejected people like them.

Sean Landis, author of *Agile Hiring* [\[Landis2011\]](#), believes that

"accepted hiring wisdom is not very effective when applied to software professionals." He further states that:

- very few companies hire well.
- individuals with deep domain knowledge are in the best position to perform great hiring.
- companies often focus on the wrong candidates.
- It is important to track metrics on the cost and effectiveness of hiring practices

In short, hiring is one of the most important decisions the digital enterprise makes, and it cannot be reduced to a simple process to be executed mechanically. Requiring senior technical talent to interview candidates may result in improved hiring decisions. However, such requirements add to the overall demand on these individuals.

Allocation and tracking people's time

When a new hire enters your organization, they enter a complex system that will structure and direct their daily activities through a myriad of means. The various means that direct their action include:

- Team assignment (e.g. to an ongoing product)
- Project assignment
- Process responsibilities

Notice again the appearance of the "3 Ps."

Product, project, and process become challenging when they are all allowed to generate demand on individuals independently of each other. In the worst case scenario, the **same** individual winds up with:

- Collaborative team responsibilities
- "Fractional" allocation to one or more projects
- Ticketed process responsibilities

NOTE

Fractional allocation is the practice of funding individuals through assigning them at some % to a project. For example, a server engineer might be allocated 25% time to a project for 6 months to define its infrastructure architecture, while being assigned 30% to another project to refresh obsolete infrastructure.

When demand is un-coordinated, these multiple channels can result in multi-tasking and dramatic overburden, and in the worst case the individual becomes the constraint to enterprise value. Project managers expect deliverables on time, and have no visibility to operational concerns (e.g. outages) that may affect the ability of staff to deliver. Ad-hoc requests "smaller than a project, bigger than a ticket" further complicate matters.

The *Phoenix Project* presents an effective and realistic dramatization of the resulting challenges. Work is entering the system through multiple channels and the overburden on key individuals (such as Brent, the lead systems engineer) has reached crisis proportions. Through a variety of mechanisms, they take control of the demand channels and greatly improve the organization's success. One of the most important lessons is well articulated by Erik, the mentor:

"Your job as VP of IT Operations is to ensure the fast, predictable, and uninterrupted flow of planned work that delivers value to the business while minimizing the impact and disruption of unplanned work, so you can provide stable, predictable, and secure IT service...You must figure out how to control the release of work into IT Operations and, more importantly, ensure that your most constrained resources are doing only the work that serves the goal of the entire system, not just one silo. [Kim2013], p.91

In order to understand the work, measuring the consumption of people's time is important. There are various time tracking approaches:

- Simple allocation of staff payroll to product or organizational line
- Project management systems (sometimes these are used for weekly time tracking, even for staff that are not assigned to projects - in such cases, placeholder operational projects are created)
- Human Resource Management systems
- Ticketing/workflow systems - advanced systems, such as those found in the Professional Services Automation sector, track time when tickets are in an "open" status.
- Backlog management systems (that may seem similar to ticketing systems)
- Home built systems

There is little industry consensus on best practices here. There are reasonable concerns about the burden of time tracking on employees, and poor data quality resulting from employees attempting to "code" activities when summarizing their time on a weekly or bi-weekly basis.

Need reviewer input

Accountability and performance

Accountability helps people use their autonomy judiciously.

— Sriram Narayam, Agile IT Organization Design

Regardless of whether the company is a modern digital enterprise or more traditional in its approach, the commitment, performance, and results of employees is a critical concern. The traditional approach to managing this has been an annual review cycle, resulting in a performance ranking from 1-5:

1. Did not meet expectations
2. Partially met expectations
3. Met expectations
4. Exceeded expectation
5. Significantly exceeded expectations

This annual rating determines the employee's compensation and career prospects in the

organization. Some companies (notably GE and Microsoft) have attempted "stack rankings" in which the "bottom" performers must be terminated. High profile practitioners however are moving away from this practice [Brustein2013], [Olson2013].

The traditional annual review is a large "batch" of [feedback](#) to the employee, and therefore ineffective in terms of systems theory, not much better than an [open-loop](#) approach. Because of the weaknesses of such slow feedback (not to mention the large annual costs, expensive infrastructure, and opportunity costs of the time spent), companies are experimenting with other approaches.

Deloitte Consulting, as reported in the Harvard Business Review [Buckingham2015], realized that its annual performance review process was consuming two million hours of time annually, and yet was not delivering the needed value. In particular, ratings were suffering from the measurable flaw that they tended to reveal more about the person **doing** the rating, than the person being rated!

They started by redefining the goals of the performance management system: to accurately identify and reward performance, and fuel its further improvements.

A new approach with greater statistical validity was implemented, based on four key questions:

- Given what I know of this person's performance, and if it were my money, I would award this person the highest possible compensation increase and bonus
- Given what I know of this person's performance, I would always want him or her on my team
- This person is at risk for low performance
- This person is ready for promotion today

In terms of the frequency of performance check-ins, they note:

Research into the practices of the best team leaders reveals that they conduct regular check-ins with each team member about near term work . . . to set expectations for the upcoming week, review priorities, comment on recent work, and provide course correction, coaching, or important new information . . .

\. . . If a leader checks in less often than once a week, the team member's priorities may become vague . . . the conversation will shift from coaching for near term work to giving feedback about past performance.

\. . . If you want people to talk about how to do their best work in the near future, they need to talk often. And so far we have found in our testing a direct and measurable correlation between the frequency of these conversations and the engagement of team members . . .

Sutton and Rao, in *Scaling up Excellence*, discuss the similar case of Adobe. At Adobe, "annual reviews required 80,000 hours of time from the 2,000 managers at Adobe each year, the equivalent of 40 full-time employees. After all that effort, internal surveys revealed that employees felt less inspired and motivated afterwards—and turnover increased."

Because of such costs and poor results, Adobe scrapped the entire performance management system in favor of a "check-in" approach. In this approach, managers are expected to have regular conversations about performance with employees, and are given much more say in salaries and

merit increases. The managers themselves are evaluated through random "pulse surveys" that measure how well each manager "sets expectations, gives and receives feedback, and helps people with their growth and development." [\[Sutton2014\]](#), p. 113.

Whether incentives (e.g. pay raises) should be awarded individually or on a team basis is an ongoing topic of discussion in the industry. Results often derive from team performance, and the contributions of any one individual can be difficult to identify. Because of this, Scrum pioneer Ken Schwaber argues that "The majority of the enterprise's bonus and incentive funds need to be allocated based on the team's performance rather than the individual's performance." [\[Schwaber2007\]](#), p.6.

However, this runs into another problem: that of the "free-rider." What do we do about team members who do not pull their weight? Even in self-organizing teams, confronting someone about their behavior is not something people do willingly, or well.

Ideally, teams will self-police, but this becomes less effective with scale. In one case study in the Harvard Business Review, Continental Airlines found that the free rider problem was less of a concern when metrics were clearly correlated with team activity. In their case, the efforts and cooperation of gate teams had significant influence on On-Time Arrival and Departure metrics, which could then be used as the basis for incentives [\[Knez2002\]](#).

Ultimately, both individuals and teams need coaching and direction. Team-level management and incentives must still be supplemented with some feedback loops centering on the individual. Perhaps this feedback is not compensation-based, but the organization must still identify individuals with leadership potential and deal with free riders and toxic individuals.

Observed behaviors are a useful focus. Sean Landis describes the difference between behaviors and skills thus:

Two things make good leaders: behaviors and skills. If you focus on behaviors in your hiring of developers, they will be predisposed for leadership success. The hired candidate may walk in the door with the skills necessary to lead or not. If not, skills are easy to acquire through training and mentoring. People can acquire or modify behaviors, but it is much harder than skill development. Hire for behaviors and train the leadership skills. [\[Landis2011\]](#)

He further provides many examples of behaviors, such as:

- Adaptable
- Accountable
- Initiative Taker
- Optimistic
- Relational

In conclusion - Many executives and military leaders have identified the central importance of hiring decisions over the years. In large, complex organizations, choosing the right people is the most powerful lever a leader has to drive organizational performance. As we discussed in Chapter 7 and in this chapter, the organizational context these new hires find themselves in will profoundly

affect them and the results of their efforts.

incorporate Carol Dweck

IT financial management

Collaborative

Status: This section is first draft complete as of 10 May 2016. Reviews and comments appreciated. Some minor changes may be seen as I review over next few days.

Needs: IT financial management experts also familiar with Beyond Budgeting or Lean Accounting. All 3 of you.

- Collaborator instructions
- Raise an issue to comment
- Github source files

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

2nd draft: add capex/opex, capitalization in Agile

2 approaches

1. Capitalize after 1st MVP (conservative)
2. capitalize all the sprints

Feasibility is key criteria

Story point capitalization - also as alternative to time tracking - see also youtube, slideshare for <https://agile2016.sched.org/event/6ebi/agile-economics-budgets-contracts-capitalization-pavel-dabrytski?iframe=yes&w=i:0;&sidebar=yes&bg=no> pavel dabrytski talk

but traditionally requirements are not capitalized

Dan Greening talk on capitalization, Agile 2016 - couldn't attend, find video

Computers initially were used to automate manual operations and the benefits were relatively easy to forecast. As organizations use computers more for strategic purposes and management information enhancements, the benefits become harder to forecast. In a growing number of companies the individuals most qualified to forecast the most significant costs and benefits are product managers, financial specialists, marketing specialists, and not the I/S technical specialists . . .

— Terence Quinlan, IT Financial Management Association

Financial health is an essential dimension of business health. And digital technology has been one of the fastest-growing budget items in modern enterprises. Its relationship to enterprise financial management has been a concern since the first computers were acquired for business purposes.

IMPORTANT

Financial management is a broad, complex, and evolving topic, and its relationship to IT and digital management is even more so. This brief section can only cover a few basics. However, it is important for you to have an understanding of the intersection of Agile and Lean IT with finance, as your organization's financial management approach can determine the effectiveness of your digital strategy. See the cited sources for further information.

The objectives of IT finance include:

- Providing a framework for tracking and accounting for digital income and expenses
- Supporting financial analysis of digital strategies (business models and operating models, including sourcing)
- Supporting the digital and IT-related aspects of the corporate budgetary and reporting processes, including internal and external policy compliance
- Supporting (where appropriate) internal cost recovery from business units consuming digital services
- Supporting accurate and transparent comparison of IT financial performance to peers and market offerings (benchmarking)

A company scaling up today would often make different decisions from a company that scaled up 40 years ago. This is especially apparent in the matter of how to finance digital/IT systems development and operations. The intent of this section is to explore both the traditional approaches to IT financial management and the emerging Agile/Lean responses.

This section has the following outline:

- Historic IT financial practices
 - Annual budgeting and project funding
 - Cost accounting and chargeback
- Next generation IT finance
 - Lean Accounting & Beyond Budgeting
 - Lean Product Development
 - Internal "venture" funding
 - Value stream orientation
 - Internal market economics
 - Service brokerage

Historic IT financial practices

Historically, IT financial management has been defined by two major practices:

- An annual budgeting cycle, in which project funding is decided
- Cost accounting, sometimes with associated internal transfers (chargebacks) as a primary tool for understanding and controlling IT expenses

Both of these practices are being challenged by Agile and Lean IT thinking.

Annual budgeting & project funding

IT organizations typically adhere to annual budgeting and planning cycles—which can involve painful rebalancing exercises across an entire portfolio of technology initiatives, as well as a sizable amount of rework and waste. This approach is anathema to companies that are seeking to deploy agile at scale. Some businesses in our research base are taking a different approach. Overall budgeting is still done yearly, but road maps and plans are revisited quarterly or monthly, and projects are reprioritized continually. [\[Comella2016\]](#)

— Comella-Dorda et al, An Operating Model for Company-Wide Agile Development

In the common practice of the annual budget cycle, companies once a year embark on a detailed planning effort to forecast revenues and how they will be spent. Much emphasis is placed on the accuracy of such forecasts, despite its near-impossibility. (If estimating one large software project is challenging, how much more challenging to estimate an entire enterprise's income and expenditures?)

The annual budget has two primary components: capital expenditures and operating expenditures, commonly called CapEx and OpEx. The rules about what must go where are fairly rigid and

determined by accounting standards with some leeway for the organization's preferences.

Software development can be capitalized, as it is an investment from which the company hopes to benefit from in the future. Operations is typically expensed. Capitalized activities may be accounted for over multiple years (therefore becoming a reasonable candidate for financing and multi-year amortization.) Expensed activities must be accounted for in-year.

Because one can only "go to the well" once a year, extensive planning and negotiation traditionally take place around the IT project portfolio. Business units and their IT partners debate the priorities for the capital budget, assess resources, and finalize a list of investments. Project managers are identified and tasked with marshaling the needed resources for execution.

This annual cycle receives much criticism in the Agile and Lean communities. From a Lean perspective, projects can be equated to large "batches" of work. Using annual projects as a basis for investment can result in misguided attempts to plan large batches of complex work in great detail, so that resource requirements can be known well in advance. The history of the Agile movement is in many ways a challenge and correction of this thinking, as we have discussed throughout this book.

The execution model for digital/IT acquisition adds further complexity. Traditionally, project management has been the major funding vehicle for capital investments, distinct from operational expense. But with the rise of Cloud computing and product-centric management, companies are moving away from traditional capital projects. New products are created with greater agility, in response to market need, and without the large capital investments of the past in physical hardware.

This does not mean that traditional accounting distinctions between capex and opex go away. Even with expensed Cloud infrastructure services, software development may still be capitalized, as may software licenses.

Cost accounting and chargeback

IT is often consumed as a "shared service" which requires internal financial transfers. What does this mean?

Here is a traditional example. An IT group purchases a mainframe computer for \$1,000,000. This mainframe is made available to various departments who are charged for it. Because the mainframe is a shared resource, it can run multiple workloads simultaneously. For the year, we see the following usage:

- 30% Accounting
- 40% Sales operations
- 30% Supply chain

In the simplest direct allocation model, the departments would pay for the portion of the mainframe that they used. But things always are more complex than that.

- What if the mainframe has excess capacity? Who pays for that?
- What if Sales Operations stops using the mainframe? Do Accounting and Supply Chain have to

make up the loss? What if Accounting decides to stop using it because of the price increase? (In public utilities, this is known as a [death spiral](#).)

- The mainframe requires power, floor space, and cooling. How are these incorporated into the departmental charges?

Historically, **cost accounting** has been the basis for much IT financial management (see e.g. ITIL Service Strategy, [\[TSO2011a\]](#), p.202; [\[Quinlan2003\]](#)). Such approaches traditionally seek *full absorption* of unit costs; that is, each "unit" of inventory ideally represents the total cost of all its inputs: materials, labor, and overhead such as machinery and buildings.

NOTE

Cost accounting is defined as "the techniques for determining the costs of products, processes, projects, etc. in order to report the correct amounts on the financial statements, and assisting management in making decisions and in the planning and control of an organization . . . For example, cost accounting is used to compute the unit cost of a manufacturer's products in order to report the cost of inventory on its balance sheet and the cost of goods sold on its income statement. This is achieved with techniques such as the allocation of manufacturing overhead costs and through the use of process costing, operations costing, and job-order costing systems. [\[Accounting2016\]](#)

In IT/digital service organizations, there are three basic sources of cost: "cells, atoms, and bits." That is:

- People (i.e. their time)
- Hardware
- Software

However, these are "direct" costs - costs that for example a product or project manager can see in their full amount.

Another class of cost is "indirect." The IT service might be charged \$300 per square foot for data center space. This provides access to rackspace, power, cooling, security, and so forth. This charge represents the bills the Facilities organization in turn receives from power companies, mechanicals vendors, security services, and so forth - not to mention the mortgage!

Finally, the service may depend on other services. Perhaps instead of a dedicated database server, the service subscribes to a database **service** that gives them a high-performance relational database, but where they do not pay directly for the hardware, software, and support services the database is based on. Just to make things more complicated, the services might be external (public Cloud) or internal (private Cloud or other offerings).

Those are the major classes of cost. But how do we understand the "unit of inventory" in an IT services context? A variety of concepts can be used, depending on the service in question:

- Transactions
- Users

- Network ports
- Storage (e.g. gigabytes of disk)

In internal IT organizations (see "[Defining consumer, customer, and sponsor](#)") this cost accounting is then used to transfer funds from the budgets of consuming organizations to the IT budget. Sometimes this is done via simple allocations (Marketing pays 30%, Sales pays 25%, etc) and sometimes this is done through more sophisticated approaches, such as defining unit costs for services.

For example, the fully absorbed unit cost for a customer account transaction might be determined to be \$0.25; this ideally represents the total cost of the hardware, software, and staff divided by the expected transactions. Establishing the models for such costs, and then tracking them, can be a complex undertaking, requiring correspondingly complex systems.

IT managers have known for years that overly detailed cost accounting approaches can result in consuming large fractions of IT resources. As AT&T financial manager John McAdam noted:

"Utilizing an excessive amount of resources to capture data takes away resources that could be used more productively to meet other customer needs. Internal processing for IT is typically 30-40% of the workload. Excessive data capturing only adds to this overhead cost." [\[McAdam2003\]](#)

There is also the problem that unit costing of this nature creates false expectations. Establishing an internal service pricing scheme implies that if the utilization of the service declines, so should the related billings. But if

1. the hardware, software, and staff costs are already sunk, or relatively inflexible and
2. the IT organization is seeking to fully recover costs

the per-transaction cost will simply have to increase if the number of transactions goes down.

James R. Huntzinger discusses the problem of excess capacity distorting unit costs, and states "it is an absolutely necessary element of accurate representation of the facts of production that some provisions be made for keeping the cost of wasted time and resources separate from normal costs" [\[Huntzinger2007\]](#). Approaches for doing this will be discussed below.

Next generation IT finance

What accounting should do is produce an unadulterated mirror of the business — an uncompromisable truth on which everyone can rely. ... Only an informed team, after all, is truly capable of making intelligent decisions.

— Orest Fiume and Jean Cunningham, as quoted by James Huntzinger

As digital transformation accelerates and companies turn to Agile operating models, criticisms of traditional approaches to IT finance have increased. Rami Sirkiä and Maarit Laanti (in a paper used as the basis for the Scaled Agile Framework's financial module) describe the following shortcomings:

- Long planning horizons, detailed cost estimates that must be frequently updated

- Emphasis on planning accuracy and variance analysis
- Context-free concern over budget overruns (even if a product is succeeding in the market, variances are viewed unfavorably)
- Bureaucratic re-approval processes for project changes
- Inflexible and slow resource re-allocation [Sirkia2013]

What do critics of cost accounting, allocated chargebacks, and large batch project funding suggest as alternatives to the historic approaches? There are some limitations evident in many discussions of Lean Accounting, notably an emphasis on manufactured goods. However, a variety of themes and approaches have emerged relevant to IT services, that we will discuss below:

- Beyond Budgeting
- Internal "venture" funding
- Value stream orientation
- Lean Accounting
- Lean Product Development
- Internal market economics
- Service brokerage

Beyond Budgeting

Setting a numerical target and controlling performance against it is the foundation stone of the budget contract in today's organization. But, as a concept, it is fundamentally flawed. It is like telling a motor racing driver to achieve an exact time for each lap . . . it cannot predict and control extraneous factors, any one of which could render the target totally meaningless. Nor does it help to build the capability to respond quickly to new situations. But, above all, **it doesn't teach people how to win.**

— Jeremy Hope and Robin Fraser, *Beyond Budgeting Questions and Answers*

Beyond Budgeting is the name of a 2003 book by Jeremy Hope and Robin Fraser. It written in part as an outcome of meetings and discussions between a number of large, mostly European firms dissatisfied with traditional budgeting approaches. Beyond Budgeting's goals are described as:

releasing capable people from the chains of the top-down performance contract and enabling them to use the knowledge resources of the organization to satisfy customers profitably and consistently beat the competition

In particular, Beyond Budgeting critiques the concept of the "budget contract." A simple "budget" is merely a "financial view of the future . . . [a] *most likely outcome* given known information at the time . . ." A "budget contract" by comparison is intended to "delegate the accountability for achieving agreed outcomes to divisional, functional, and departmental managers." It includes concerns and mechanisms such as

- Targets
- Rewards
- Plans
- Resources
- Coordination
- Reporting

and is intended to "commit a subordinate or team to achieving an agreed outcome."

Beyond Budgeting identifies various fallacies with this approach, including:

- The idea that fixed financial targets maximize profit potential
- Financial incentives build motivation and commitment (see discussion on [\[motivation\]](#))
- Annual plans direct actions that maximize market opportunities
- Central resource allocation optimizes efficiency
- Central coordination brings coherence
- Financial reports provide relevant information for strategic decision-making

Beyond the poor outcomes that these assumptions generate, up to 20 to 30% of senior executives' time is spent on the annual budget contract. Overall, the Beyond Budgeting view is that the budget contract is

a relic from an earlier age. It is expensive, absorbs far too much time, adds little value, and should be replaced by a more appropriate performance management model. [\[Hope2001\]](#), p 4, emphasis added.

Readers of this textbook should at this point notice that many of the Beyond Budgeting concerns reflect an Agile/Lean perspective. The fallacies of efficiency and central coordination have been discussed throughout this book. However, if an organization's financial authorities remain committed to these as operating principles, digital transformation will be difficult at best.

Beyond Budgeting proposes a number of principles for understanding and enabling organizational financial performance. These include:

- Event-driven over annual planning
- On-demand resources over centrally allocated resources
- Relative targets ("beating the competition") over fixed annual targets
- Team-based rewards over individual rewards
- Multi-faceted, multi-level, forward-looking analysis over analyzing historical variances

===== Internal "venture" funding

A handful of companies are even exploring a venture-capital-style budgeting model. Initial funding is provided for minimally viable products (MVPs), which can be released quickly, refined according to customer feedback, and relaunched in the marketplace . . . And subsequent funding is based on how those MVPs perform in the market. Under this model, companies can reduce the risk that a project will fail, since MVPs are continually monitored and development tasks reprioritized . . .

[Comella2016]

— Comella-Dorda et al, An Operating Model for Company-Wide Agile Development

As we have discussed [previously](#), product and project management are distinct. Product management in particular has more concern for overall business outcomes. If we take this to a logical conclusion, the product portfolio becomes a form of investment portfolio, managed not in terms of schedule and resources, but rather in terms of business results.

This implies the need for some form of internal venture funding model, to cover the initial investment in a minimum viable product. If and when this internal investment bears fruit, it may become the basis for a value stream organization, which can then serve as a vehicle for direct costs and an internal services market (see below). McKinsey reports the following case:

A large European insurance provider restructured its budgeting processes so that each product domain is assigned a share of the annual budget, to be utilized by chief product owners. (Part of the budget is also reserved for requisite maintenance costs.) Budget responsibilities have been divided into three categories: a development council consisting of business and IT managers meets monthly to make go/no-go decisions on initiatives. Chief product owners are charged with the tactical allocation of funds—making quick decisions in the case of a new business opportunity, for instance—and they meet continually to rebalance allocations.

Meanwhile, product owners are responsible for ensuring execution of software development tasks within 40-hour work windows and for managing maintenance tasks and backlogs; these, too, are reviewed on a rolling basis. As a result of this shift in approach, the company has increased its budgeting flexibility and significantly improved market response times.

[Comella2016]

With a rolling backlog and stable funding that decouples annual allocation from ongoing execution, the venture-funded product paradigm is likely to continue growing. A product management mindset activates a variety of useful practices, as we will discuss in the next section.

Options as a portfolio strategy

In governing for effectiveness and innovation, one technique is that of options. Related to the idea of options is parallel development.

In investing terms, purchasing an option gives one the right, but not the obligation, to purchase a stock (or other value) for a given price at a given time. Options are an important tool for investors to develop strategies to compensate for market uncertainty.

What does this have to do with developing digital products?

Product development is so uncertain that sometimes it makes sense to try several approaches at once. This in fact was how the program to develop the first atomic bomb was successfully managed.

Parallel development is analogous to an options strategy. Small, sustained investments in different development "options" can optimize development payoff in certain situations.

Taken to a logical conclusion, such an options strategy starts to resemble the portfolio management approaches of venture capitalists. Venture capitalists invest in a broad variety of opportunities, knowing that most in fact will not pan out. See discussion of [internal venture funding](#) as a business model.

It is arguable that the venture-funded model has created different attitudes and expectations towards governance in West Coast "unicorn" culture. However, it would be dangerous to assume that this model is universally applicable. A firm is more than a collection of independent sub-organizations; this is an important line of thought in management theory, starting with Coase's "The Nature of the Firm" [\[Coase1937\]](#).

cite Anderson's relevant work

Lean Product Development

Because we never show it on our balance sheet, we do not think of [Design in Process] as an asset to be managed, and we do not manage it.

— Don Reinertsen, Managing the Design Factory

The [Lean Product Development](#) thought of Don Reinertsen was discussed extensively in Chapter 4. His emphasis on employing an economic framework for decisionmaking is relevant to this discussion as well. In particular, his concept of [Cost of Delay](#) is poorly addressed in much IT financial planning, with its concerns for full utilization, efficiency, and variance analysis. Other Lean accounting thinkers share this concern, e.g.:

the cost-management system in a lean environment must be more reflective of the physical operation. It must not be confined to monetary measures but must also include nonfinancial measures, such as quality and throughput times. [\[Huntzinger2007\]](#)

Another useful Reinertsen concept is that of **design in process** or DIP. This is an explicit analog to the well-known Lean concept of **work in process** (WIP). Reinertsen makes the following points [\[Reinertsen1997\]](#), p13:

- DIP is larger and more expensive to hold than WIP
- It has much lower turn rates

- It has much higher holding costs (e.g. due to obsolescence)
- DIP's liabilities are ignored due to weaknesses in accounting standards

These concerns give powerful economic incentives for managing throughput and flow, continuously re-prioritizing for maximum economic benefit and driving towards the Lean ideal of single-piece flow.

Lean Accounting

It was not enough to chase out the cost accountants from the plants, the problem was to chase cost accounting from my people's minds.

— Taiichi Ohno

There are several often-cited motivations for cost accounting ([\[Huntzinger2007\]](#) p 13):

- Inventory valuation (not applicable for intangible services)
- Informing pricing strategy
- Management of production costs

NOTE The term "cost accounting" is used here with its [full implications](#), including allocated overhead. It is not the same as just "accounting for costs," which is always a concern for any organization.

IT service costing has long presented additional challenges to traditional cost accounting. As IT Financial Management Association president Terry Quinlan notes, "Many factors have contributed to the difficulty of planning EDP expenditures at both application and overall levels. A major factor is the difficulty of measuring fixed and variable cost." [\[Quinlan2003\]](#), p. 6.

This begs the broader question: should traditional cost accounting be the primary accounting technique used at all? Cost accounting in Lean and Agile organizations is often controversial. Lean pioneer Taiichi Ohno of Toyota thought it was a flawed approach. Huntzinger [\[Huntzinger2007\]](#) identifies a variety of issues:

- Complexity
- Un-maintainability
- Supplies information "too late" (i.e., does not support fast feedback)
- "Overhead" allocations result in distortions

Shingo Prize winner Steve Bell observes:

It is usually impossible to tie . . . abstract cost allocations and the resulting variances back to the originating activities and the value they may or may not produce; thus, they can't help you improve. But they can waste your time and distract you from the activities that produce the desired outcomes . . . [\[Bell2013\]](#), p 110.

The trend in Lean Accounting has been to simplify. A guiding ideal is seen in the [Wikipedia article on Lean Accounting](#):

The "ideal" for a manufacturing company is to have only two types of transactions within the production processes; the receipt of raw materials and the shipment of finished product.

Concepts such as value stream orientation, internal market economics, and service brokering all can contribute towards achieving this ideal.

Value stream orientation

Collecting costs into traditional financial accounting categories, like labor, material, overhead, selling, distribution, and administrative, will conceal the underlying cost structure of products . . . The alternative to traditional methods . . . is the creation of an environment that moves indirect costs and allocation into direct costs. [\[Huntzinger2007\]](#)

— James R. Huntzinger, Lean Cost Management

As discussed above, Lean thinking discourages the use of any concept of overhead, sometimes disparaged as "peanut butter" costing. Rather, direct assignment of all costs to the maximum degree is encouraged, in the interest of accounting simplicity.

We discussed a venture-funded product model above, as an alternative to project-centric approaches. Once a product has proven its viability and becomes an operational concern, it becomes the primary vehicle for those concerns previously handled through cost accounting and chargeback. The term for a product that has reached this stage is "value stream." As Huntzinger notes, "Lean principles demand that companies align their operations around products and not processes." [\[Huntzinger2007\]](#) p 19.

By combining a value stream orientation in conjunction with organizational principles such as frugality, internal market economics, and decentralized decision-making (see e.g. [\[Hope2001\]](#) p 12), both Lean and Beyond Budgeting argue that more customer-satisfying and profitable results will ensue. The fact that the product in this case is digital (not manufactured), and the value stream centers around product development (not production) does not change the argument.

Internal market economics

value stream and product line managers, like so much in the lean world, are "fractal."

— Womack and Jones, Lean Thinking

Coordinate cross-company interactions through “market-like” forces.

— Jeremy Hope and Robin Fraser, Beyond Budgeting Questions and Answers

IT has long been viewed as a "business within a business." In the internal market model, services consume other services, ad infinitum [\[Meyer2013\]](#). Sometimes the relationship is hierarchical (an

application team consuming infrastructure services) and sometimes it is peer to peer (an application team consuming another's services, or a network support team consuming email services, which in turn require network services.)

The increasing sourcing options including various Cloud options make it more and important that internal digital services be comparable to external markets. This in turn puts constraints on traditional IT cost recovery approaches, which often result in charges with no seeming relationship to reasonable market prices.

There are several reasons for this. One commonly cited reason is that internal IT costs include support services, and therefore cannot fairly be compared to simple retail prices (e.g. for a computer as a good.)

Another, more insidious reason is the rolling in of unrelated IT overhead to product prices. We have quoted James Huntzinger's work above in various places on this topic. Dean Meyer has elaborated this topic in greater depth from an IT financial management perspective, calling for some organizational "goods" to be funded as either Ventures (similar to above discussion) or "subsidies" (for enterprise-wide benefits such as technical standardization) [\[Meyer2013\]](#), p.92.

As discussed above, a particularly challenging form of IT overhead is excess capacity. The saying "the first person on the bus has to buy the bus" is often used in IT shared services, but is problematic. A new, venture-funded startup cannot operate this way - expecting the first few customers to fully fund the investment! Nor can this work in an internal market, unless heavy handed political pressure is brought to bear. This is where internal venture funding is required.

Meyer presents a sophisticated framework for understanding and managing an internal market of digital services. This is not a simple undertaking; for example, correctly setting service prices can be surprisingly complex.

Service brokerage

Finally, there is the idea that digital or IT services should be aggregated and "brokered" by some party (perhaps the descendant of a traditional IT organization). In particular, this helps with capacity management, which can be a troublesome source of internal pricing distortions. This has been seen not only in IT services, but in Lean attention to manufacturing; when unused capacity is figured into product cost accounting, distortions occur [\[Huntzinger2007\]](#), chapter 7, "Church and Excess Capacity."

Applying Meyer's principles, excess capacity would be identified as a Subsidy or a Venture as a distinct line item.

But Cloud services can assist even further. Excess capacity often results from the available quantities in the market - e.g., one purchases hardware in large-grained capital units. But more flexibly priced, expensed compute on demand services are available, it is feasible to allocate and de-allocate capacity on demand, eliminating the need for accounting for excess capacity.

2nd draft material
spotify DIBB framework
===== Allocation of capital in large batches

Pros and cons - sometimes, one needs to accept the fog of R&D
This never changes
The iconic entrepreneur asking for one more round

entrepreneurs tell a story <https://silvialisam.com/chris-sacca-s-639f0f5f3b9c#.j6sljdp0x>

VC view

review Lean Enterprise ch13

Topics in IT project management and related fields

Collaborative

Status: Like most topics sections, this one is in need of work, being deferred to 2nd draft for cleanup & completion.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Investment management?

The Agile project frameworks

As of this writing, a number of frameworks have been developed at the intersection of Agile and project management. Notable examples include:

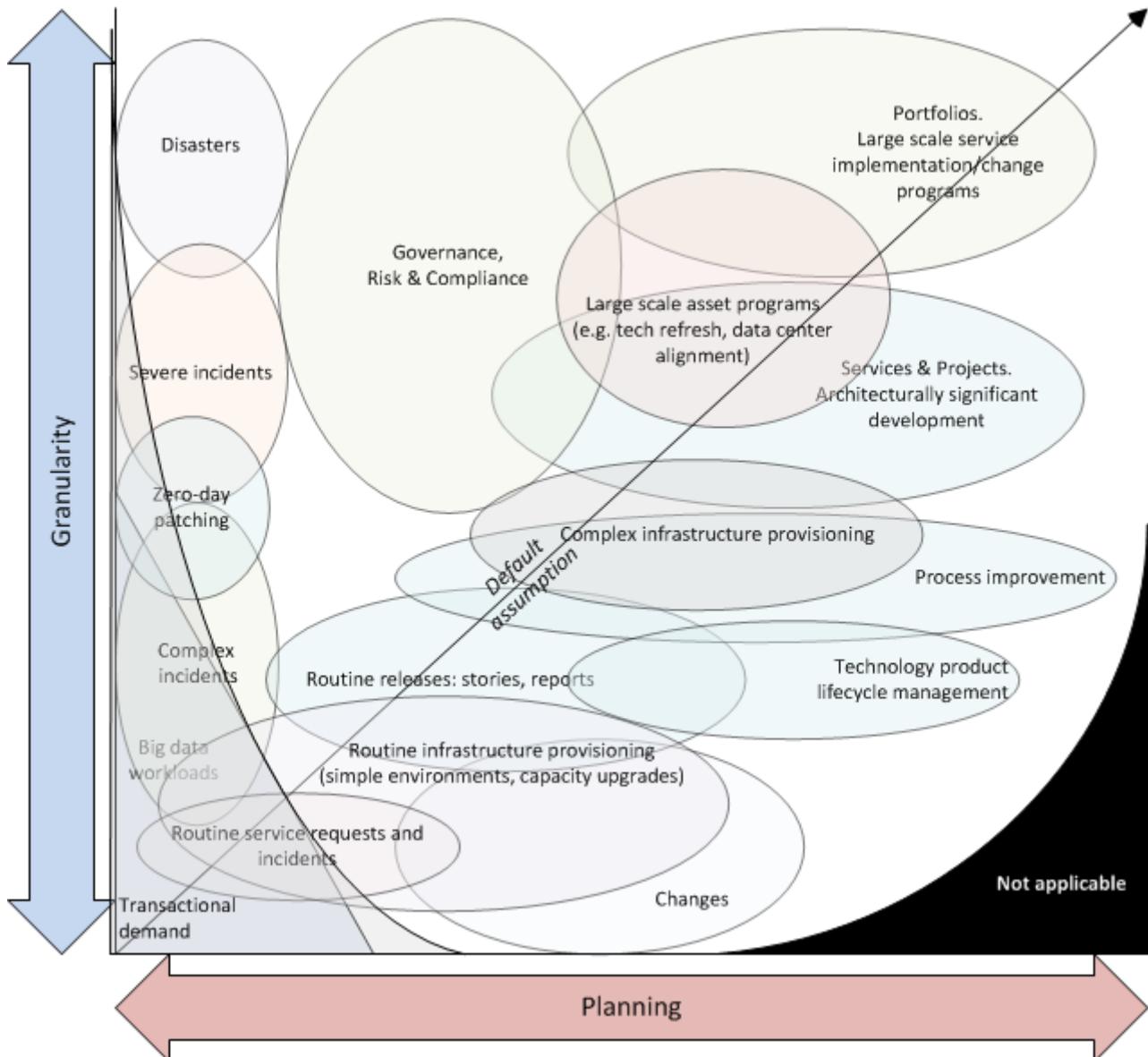
- The Scaled Agile Framework (<http://www.scaledagileframework.com/>)
- Disciplined Agile Delivery (<http://www.disciplinedagiledelivery.com/>)

expand with contributed descriptions

Other Agile authors are skeptical of the need for such material [Schwaber2013].

The two dimensions of demand management

Project management is our first major specialization of Chapter 5's general concept of "work management." (We will turn to process management in the next chapter.) In the interest of keeping project management firmly in context, consider the following two-dimensional analysis:



You should spend some time considering the graphic, which provides a unique way of understanding the work you are now experiencing as a "team of teams" or "manager of managers" in an IT-dependent environment of increasing size and complexity.

We've come a long ways from the last section's insistence that "it's all just work." But there still is an essential truth there. Regardless of the abstract label we put on a given activity, it represents some set of tasks or objectives that real people are going to take time to perform, and expect to be compensated for. Retaining a clear realization of this is essential to maintaining truly business-

aligned IT management.

Let's consider the various forms that demand may take. While this may not be project management per se, it's essential to putting project management in a broader context. This will also help you develop a deeper understanding of an **architecture of IT management**, a topic the author has explored in other works [Betz2011].

The diagram has two dimensions:

- Planning
- Granularity

The bubbles represent a "space" where one is likely to find that kind of work. As indicated by the central diagonal, it reflects an assumption that larger amounts of work are more likely to be planned further in advance. However, this is not always true. A large, unwelcome amount of required work that shows up with no planning is probably a disaster. Desired work (in the form of aggregate transactional demand) may also spike unexpectedly.

[The following table is work in process, currently with just a few examples filled in]

Type of work	Description
Core transactional demand	This is the demand on the fully automated IT system for a given moment of truth: a banking account lookup, a streaming movie, a Human Resources record update
Routine service requests and incidents	Service requests are predefined, process-driven work items, rarely requiring creative thought or analysis. Incidents span a spectrum, but some are simpler and more routine than others, especially those stemming from user misunderstanding or error.
Changes	Changes represent modifications of established IT functionality or state . They represent some definite risk to one or more IT services, which is why they are planned on a longer lead time. However, they ideally remain relatively granular, which helps reduce their risk.
Routine releases, stories, reports	Releases and (in the Agile world) stories represent larger increments of functionality
Services and Projects	A Project is a large, planned amount of work with a defined end date. It might create a Service, which also represents a commitment to a large, ongoing amount of work, perhaps comparable in scope to the Project.

Type of work	Description
Severe incidents	Severe incidents by definition are not planned. But they represent a significant amount of work to overcome.

Some forms of work may lead to other forms of work. For example, Projects may manifest as Stories, Releases, and Changes. This complicates the diagram a bit; we don't want to "double-count" work effort. But not all Releases derive from Projects, and not all Project work (especially in complex environments) can be cleanly reduced to a set of smaller tasks.

The final point of this diagram: you only have so much capacity to execute the work it implies. If you have a disaster, it may impact your ability to deliver user stories, changes, or even meet transactional demand. Trade-offs must be considered.

Controversies in PM history

PERT chart history, McNamara, etc

The estimation controversy

<http://synerzip.com/must-see-agile-tv/to-estimate-or-no-estimates-that-is-the-question/>
<http://www.jrothman.com/mpd/project-management/2016/06/the-case-for-and-against-estimates-part-1/> <http://www.jrothman.com/mpd/project-management/2016/06/the-case-for-and-against-estimates-part-2/>

Earned value management

EVM could be described as a "model" used for understanding the execution of software projects. It includes mathematical formulae which are intended to provide insights into project dynamics. However as will be discussed, the model does not take into consideration feedback. It is an Open-Loop system model. Kennaley p44

or translating EVM directly to the balance sheet (EVM always being a fiction and less and less useful in a digitally transforming world)

The systems problem

When risk and uncertainty are present with ongoing demand, like in the case of architectural changes and related high-stake decisions, batching demand into a project for governance reasons makes sense...

— Mark Kennaley, SDLC 3.0

Kennaley, above, provides some reasoning for project management. Ironically, he uses the term "batch," which is often a concept unfavorably viewed in Agile and Lean circles.

Not all problems respond to iteration. Sometimes, deeper analysis is necessary. When a system

presents multiple constraints that need to be solved simultaneously, the only way to do this with confidence may be as a "batch" of thought and analysis. If one needs to balance several competing objectives, one may need to know how a given course of action will affect all of them before embarking. Solving for one constraint (even iteratively), and embarking on a plan of action, may result in not meeting the other constraints, and having to throw the whole solution away.

Dogmatic approaches to solving this are unhelpful. Multi-constraint optimization is a known hard problem in systems engineering and cannot be wished away through merely invoking the word "Agile."

Project management antipatterns

- Impersonal requirements (but see Alleman)
- "nothing is real unless it's a project"
- "Water-scrum-fall"

When taken to an extreme, the PMO becomes the arbiter of ALL IT investments in some organizations. Support and operational activities are chartered as long-running "projects" and staff time is billed through them (more on this in the section on Resource Management.) Essentially, this is how a program or product management approach emerges from taking the project management paradigm to an ultimate conclusion.

Scrum of scrums and its critiques

identical to a chapter-based grassroots organization
trouble is that emergent problems at each level start to distract & take over by the time you are to level 3, how do you maintain "credibility" at the base ... old trade union problem ... the quest for "authenticity" and grassroots suspicion of "egghead" systems thinkers

The concept of "scrum of scrums" may make intuitive sense, but starts to run into issues of power, hierarchy, authority, accountability, and so forth. It is mostly presented as a bottom-up structure, and there is an implication that the representatives are also practitioners. (There is a strong narrative of authenticity in the Agile community. The lack of respect for "non-coders" is pronounced.) But what happens when overriding systems concerns need to be communicated downwards in the face of team resistance? What of the "scrum of scrums" representative who is perceived (rightly or wrongly) as not advocating reliably for the team? What of the team who rotates their delegate out of well-intentioned motives related to individual empowerment, but resulting in ineffective representation?

In these questions we see the overlap of Agile with seemingly distant fields such as political science, organizational leadership, and the like. These are old problems with no "solutions" - only satisficing techniques.

Notes/2nd draft followup

From project to product/service: do current service portfolio techniques have a robust time/risk dimension, like PPM? (Galleman twitter discussion 7/4)

Project v program: Program: This term can describe two different situations. First, a program can be a set of goals that gives rise to specific projects, but, unlike a project, a program can never be completely accomplished. For example, a health-awareness program can never completely achieve its goal (the public will never be totally aware of all health issues as a result of a health-awareness program), but one or more projects may accomplish specific results related to the program's goal (such as a workshop on minimizing the risk of heart disease). Second, a program sometimes refers to a group of specified projects that achieve a common goal.

Portny, Stanley E. (2013-04-09). Project Management For Dummies (Kindle Locations 678-683). Wiley. Kindle Edition.

Tengshe, Ash, and Scott Noble. 2007. Establishing the agile PMO: Managing variability across projects and portfolios. In Proceedings of the Agile 2007 Conference, ed.Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 188-193. IEEE Computer Society.

Tengshe and Noble established the agile project management office at Capital One Auto Finance. This paper describes their experience doing so and provides good advice for transitioning a PMO from traditional to agile. (cited in [Cohn2010])

<http://www.disciplinedagiledelivery.com/agility-at-scale/geographically-distributed-agile-teams/>

Investment scorecarding pseudoscience - hubbard (example in Bente p91)

The concept of last responsible moment Systems and automation Skills management-10x controversy Processes within projects-risk, action, issue, etc

idea: interview St. Thomas IT staff on moving to Office 365 (resource angle). The modern product manager: a purple squirrel? [Cohn2010]pp130-132

Centers of Excellence as resource governors... Allocating limited resources (money, people, etc.) across all their possible use is an important function of CoEs. They should ensure organizations invest in the most valuable projects and create economies of scale for their service offering. In addition, coordination across other corporate interests is needed to enable the CoE to deliver value (from Trebec_ITAMFromTheTrenches.pdf)

How does the COE know how to prioritize?

<https://beta-get.aptage.com/wp-content/uploads/2016/04/The-Aptage-Release-Planner-V3.pdf>
<http://blackswanfarming.com/fund-the-teams-not-the-project/>

A holistic product-oriented approach contradicts a convenient oversimplification of IT delivery where everything is treated as a project. Such an approach is not Lean, in that treating all demand (coarse/strategic demand the same as fine-grained IT demand) amounts to batching, which creates inventory, waiting and potentially over-processing types of waste.

— Mark Kennaley, SDLC 3.0

When this chapter is refactored to Investment, forward refs to Chapter 12 Portfolio Mgmt.

===== Portfolio Cost of Delay

portfolio-level xref:portfolio-CoD[Cost of Delay] as essential to defining execution management. <>Reinertsen1997>> pp.35-38.

Conclusion

fundamental theme - what can we know... emergence vs planning ... network vs chain ... finding partial ordering, falsifying hypotheses...

If we "get rid of the PMO" there still is the need to charter new investments. What is their value proposition, business model... there is always a portfolio and some form of success criteria and there will ALWAYS be issues of resources and dependencies

Discussion questions

- Imagine yourself in an organization that recognized product management and work management, but had no concept of project management. Describe a scenario where project management would be a reasonable technique to introduce.
- What experience do you or your team members have with project management? How effective did you find it?
- Do microservices/continuous delivery/devops render traditional PM obsolete?

Research & practice

- Compare Microsoft Project with one of the following. What are the pros and cons?
 - Rally
 - Jira
 - VersionOne
 - Asana

Further reading

Project management is a deep and wide field, at its most mature showing the influence of operations theory and research. This section is intended to give you a brief overview of its foundational elements and reach. For a more in depth treatment, see [PMI2013] and [Richardson2010]. For Agile, Lean, and Theory of Constraints aspects, see [Cobb2015], [Goldratt1997], and (once again) [Reinertsen2009] (especially the discussion of Weighted Shortest Job First).

Books

- Charles G. Cobb, [The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach](#)
- Project Management Institute, [A Guide to the Project Management Body of Knowledge \(Pmbok Guide\) - 5th Edition](#)
- Gary L. Richardson, [Project Management Theory And Practice](#)
- Brooks, [Mythical Man-Month](#)

Articles

- [Project Management 101](#)
- [The Need for Agile Project Management](#)
- [Project vs. Product](#)
- [Scaling Agile: Problems and Solutions](#)
- [Agile Contracts](#) by Larman & Vodde

<https://www.infoq.com/articles/noprojects1-projects-flawed>

<https://www.infoq.com/articles/noprojects2-focus-value> <https://www.infoq.com/articles/noprojects3-value-change>

<http://www.acq.osd.mil/evm/resources/EVM-Agile%20Meeting.html>

<http://www.acq.osd.mil/evm/resources/DoDAgileSep2015.html>

Resources

Videos

Professional

Chapter 9: Process Management

Collaborative

Status: These three chapter sections are in first draft as of 4 June 2016:

Introduction

Why Process Management?

Process Management Practices

- Collaborator instructions
- Raise an issue to comment
- Github source files

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Introduction to Chapter 9

You're hiring more specialized people with specialized backgrounds. Many of them enter your organization and immediately ask process questions:

- What's your Change process?
- What's your Problem process?
- What's your Portfolio process?

You've not had these conversations before. What do they mean by these different "processes?"

They seem to have some expectation from their previous employment, and if you say "we don't have one" they tend to be confused. You're becoming concerned that your company may be running some risk, although you also are wary that "process" might mean slowing things down, and you can't afford that.

IMPORTANT

As with other chapters in the later part of this book, we are going to introduce Process Management "on its own terms." We will then add additional context and critique in subsequent sections.

What is "process?"

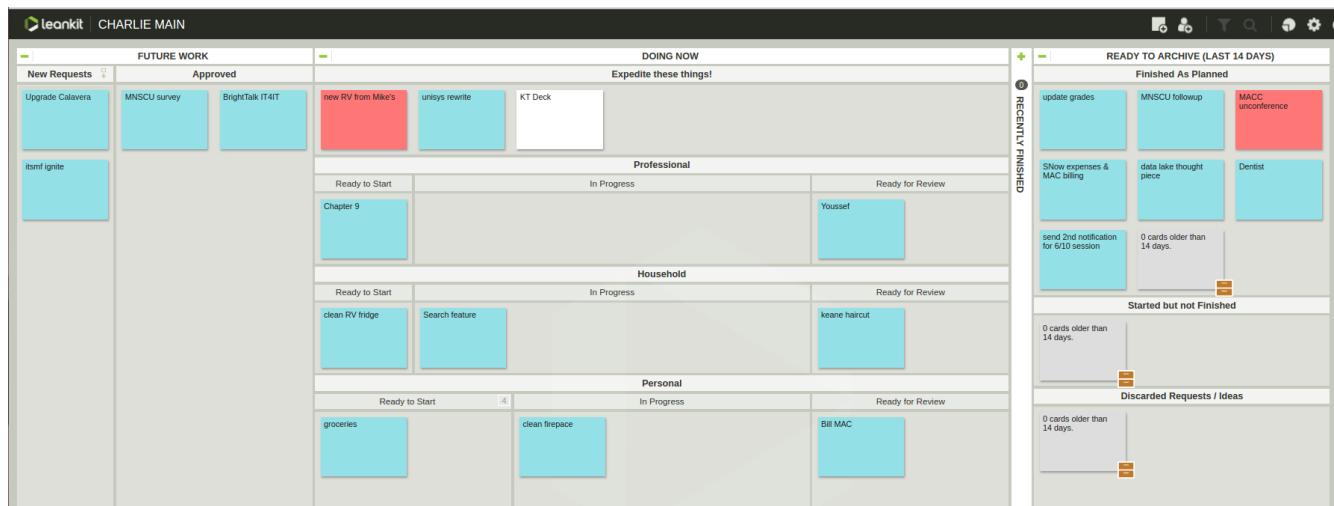
Dictionary.com defines it as "a systematic series of actions directed to some end . . . a continuous action, operation, or series of changes taking place in a definite manner."

We've discussed work management, which is an important precursor of process management.

Work management is less formalized; a wide variety of activities are handled through flexible **Kanban**-style boards or "card walls" based on the simplest "process" of:

- To do
- Doing
- Done

However, the simple card wall/Kanban board can easily become much more complex, with the addition of swimlanes and additional columns, including holding areas for blocked work:



to be redrawn

And when tasks become more routine and specialized, formal process management starts to emerge.

Processes are how we ensure consistency, repeatability, and quality. You get expected treatment at banks, coffee shops, and dentists because they follow processes. IT systems often enable processes – a mortgage origination system is based on IT software that enforces a consistent process. IT management itself follows certain processes, for all the same reasons as above.

Process, in terms of "business process," has been a topic of study and professional development for many years. Pioneering Business Process Management (BPM) authors such as Michael Hammer [Hammer1993] and Geary Rummler [Rummler1995] have had an immense impact on business operations. Toyota considers a clear process vision, or "target condition," to be the most fundamental objective in improving operations ([Rother2010], chapters 5-6). Designing processes, improving them, and using them to improve overall performance is an ongoing activity in most, if not all organizations.

However, process can cause problems. Processes imply **queues**, and in digital and other product development-based organizations, this means invisible work in process. For every employee you hire who expects you to have processes, another will have bad process experiences at previous employers. Nevertheless, process remains an important tool in your toolkit for organization design.

Process is a broad concept used throughout business operations. **The coverage here is primarily about process as applied to the digital organization.** There is a bit of a recursive paradox here;

in part, we are talking about the **process by which business processes are analyzed and automated**. By definition, this overall "process" (you could call it a **meta-process**) cannot be made too prescriptive or predictable.

We also introduce process in terms of our [emergence model](#), and therefore focus on the concerns and dynamics that may cause you to notice the concept, and your need for it.

There is a rich body of industry practice we will draw on in our discussion. Business process management for the purposes of the digital organization is just one more form of business process management.

Chapter 9 overview

In this section, we will cover:

- Why process management matters, and is not likely to go away
- The basics of process description and modeling
- Some background and history of continuous improvement and process control
- The major IT-related process frameworks
- Emerging approaches for digital process management.

This remains a textbook primarily aimed at digital product management (from concept to operations), and this chapter is primarily about business process **in that context**. However, one of the problems with process management in IT and digital management is that it has not been well aligned to the broader topic of business process management in general. Therefore, this chapter starts with discussion of BPM fundamentals, and then covers IT process management in that light.

Chapter 9 learning objectives

- Identify differences between process and project management
- Describe a business process and how it crosses functions
- Demonstrate familiarity with basic principles of process modeling
- Understand the difference between defined and empirically controlled processes, and why this is important for digital product management
- Identify the problems of process proliferation with respect to execution and demand
- Describe the origins, purpose, and scope of IT industry frameworks

Why process management?

Is a firm a collection of activities or a set of resources and capabilities? Clearly, a firm is both. But activities are what firms do, and they define the resources and capabilities that are relevant.

— Michael Porter, Competitive Advantage

We have previously discussed product and project management. You may wish to review our

summary description of process, product, and project management.

Process management starts with the fundamental capability for coordinated work management, and refines it much further.

In our company, work has been specializing. A simple card-based kanban approach is no longer sufficient. We are finding that some work is repetitive, and we need to remember to do certain things in certain orders.

For example, a new Human Resources (HR) manager was hired. The HR manager decided that a sticky note of "hire someone new for us" was not sufficient. As she pointed out to the team, hiring needed to follow a defined sequence of tasks:

- Establishing the need and purpose for the position
- Soliciting candidates
- Filtering the candidates
- Selecting a final choice
- Registering that new employee in the payroll system
- Getting the new employee set up with benefits providers (insurance, retirement, etc)
- Getting the new employee working space, equipment, and access to systems
- Training the new employee in organizational policies, as well as any position-specific orientation

The sales, marketing, and finance teams have similarly been developing ordered lists of tasks that are consistently executed as end to end sequences. And even in the core digital development and operations teams, they are finding that some tasks are repetitive and need documentation so they are performed consistently.

More broadly, your entire digital product pipeline is sometimes called a "process." From initial feature idea through production, you seek a consistent means for identifying and implementing valuable functionality. Sometimes this work requires intensive, iterative collaboration and is unpredictable (e.g. developing a user interface); sometimes, the work is more repeatable (e.g. packaging and releasing tested functionality).

However, some team members are cautious of the word "process". The term "process police" comes up in an unhappy way.

"Are we going to have auditors tracking whether we filled out all our forms correctly?" one asks.

"We used to get these consultants in at my old company and they would leave piles of 3 ring binders on the shelf that no one ever looked at," another person says.

"I can't write code according to someone's recipe!" a third says with some passion, and to general agreement from the developers in the room.

The irony is that digital products are based on process automation. The idea that certain tasks can be done repeatably and at scale through digitization is fundamental to all use of computers. The digital service **is** fundamentally an automated process, one that can be intricate and complicated.

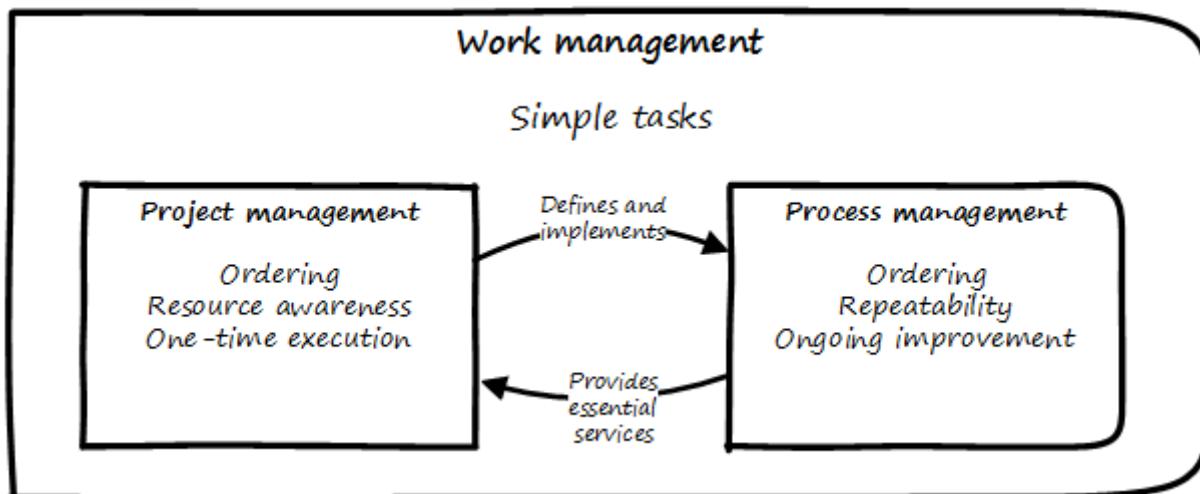
That's what computers do. But process management also spans human activities, and that's where things get more complex.

From work to process management

So, the kanban board has started to elaborate. We're seeing more and more work that needs to follow a sequence, or checklist... Wait, didn't we see that in the chapter on Project Management?

We discussed task ordering in Chapter 8. However, projects are not usually seen as repeatable.

It all starts with just "work." The following diagram might help:



Project management adds concerns of task ordering and resource management, for projects typically executed on a one-time basis.

Process management is also concerned with ordering, but less so with the resource load (more on this below), and instead is concerned with repeatability and ongoing improvement.

Process management is when we need to start managing

- multiple,
- repeatable,
- measurable sequences of activity,
- considering their interdependencies,
- perhaps using common methods to define them,
- and even common tooling to support multiple processes.

Project management and process management interact in 2 primary ways:

- Projects often are used to create and deploy processes. A large system implementation (e.g. of a Enterprise Resource Planning module such as Human Resource Management) will often be responsible for process implementation including training.
- As environments mature, product and/or project teams require process support.

As Richardson notes in *Project Management Theory and Practice*, "there are many organizational

processes that are needed to optimally support a successful project." [\[Richardson2010\]](#) For example, the project may require predictable contractor hiring, or infrastructure provisioning, or security reviews. The same is true for product teams that may not be using a "project" concept to manage their work. To the extent these are managed as repeatable, optimized processes, risk is reduced.

Process and organization

Processes are much more than repeatable activities. Many leading thinkers (such as Michael Porter, quoted at the start of this chapter section) see organizations primarily as sets of interacting processes, supporting fundamental end to end value chains or value streams. We will talk more about this in the next chapter section.

The concept of process is often contrasted with that of function or organization. We have seen in previous chapters how product development and project management must drive results across organizational boundaries. Process management has a similar challenge; its goal is to drive **repeatable** results across organizational boundaries.

As we know from our discussion of [Product Management](#), developing new products is not a particularly repeatable process. The Agile movement in some ways arose in opposition to attempts to apply process concepts of "repeatability" to developing software. These concerns remain.

However, this book is not only about digital R&D processes. It is also about a spectrum of operations and effort that spans from the unique to the highly repeatable. There is an interesting middle ground, of processes that are at least semi-repeatable. Examples often found in the large digital organization include:

- Assessing, approving, and completing changes
- End user equipment provisioning
- Resolving incidents and answering user inquiries
- Troubleshooting problems

And many others. We will talk about a wide variety of such processes in the chapter section on process frameworks.

It is important to realize, however, that just as the traditional IT project is under pressure, there are similar challenges for the traditional IT process. Continuous deployment techniques are eroding the need for formal change management. Consumerization is challenging traditional internal IT provisioning practices. And self-service help desks are eliminating some traditional support activities.

Nevertheless, any rumors of an "end to process" are probably greatly exaggerated. There will likely always be complex combinations of automated, semi-automated, and manual activity in digital organizations. Some of this activity will be repeatable enough that the "process" construct will be applied to it. Measurability remains a concern; the Lean philosophy underpinning much Agile thought emphasizes this.

It is therefore useful to understand more fundamentally what processes are, how they operate, and how they are managed and improved.

Process as skill

As we discussed at the beginning of this chapter, sometimes new employees come in expecting that you are following certain processes. This is in part because "process" experience can be an important part of an employee's career background. A skilled HR manager may consider their experience with large-scale enterprise hiring processes to be a major part of their qualifications for a position in your company.

This applies to both "business" and "IT" processes. In fact, in the digital world, there is no real difference. Digital processes:

- Initiate new systems, from idea to construction
- Publicize and grant access to the new systems
- Capture revenue from the systems
- Support people in their interactions with the systems
- Fix the systems when they break
- Improve the systems based on stakeholder feedback

It's not clear which of these are "IT" versus "business" processes. But they are definitely processes. Some of them are more predictable, some less so, but they all represent some form of ordered work that is repeatable to some degree. And to some extent, you are seeking people with experience defined at least in part by their exposure to processes.

NOTE In Chapter 10, we will discuss IT governance in depth. The concept of "control" is critical to IT governance, and processes often play an important role in terms of control.

Measuring process

One of the most important reasons for repeatable processes is so that they can be measured and understood. Repeatable processes are measured in terms of:

- Speed
- Effort
- Quality
- Variation
- Outcomes

at the most general level, and of course all of those measurements must be defined much more specifically depending on the process.

Measurement is an essential aspect of process management, but must be carefully designed. Measuring process can have unforeseen results. Process participants will behave according to how the process is measured. If a help desk operator is measured and rated on how many calls they process an hour, the quality of those interactions may suffer. It is critical that any process "key performance indicator" be understood in terms of the highest possible business objectives. Is the

objective truly to process as many calls as possible? Or is it to satisfy the customer so they need not turn to other channels to get their answers? (More on measurement in the chapter section on [Continuous Improvement](#).)

===== The core IT service management processes

Change
Service Request
Incident
Problem

There are many more hypothetical... discussed subsequently

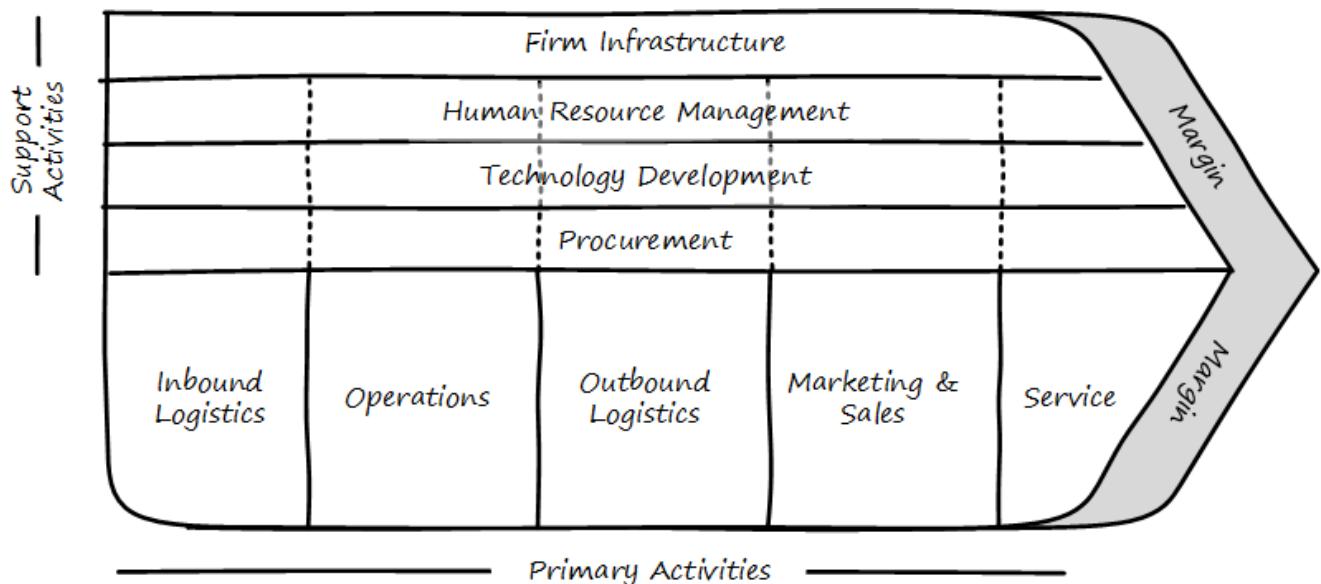
Process basics

BPM fundamentals

We've discussed some of the factors leading to the need for process management, but we haven't yet come to grips with what it **is**.

Process management is grounded in business fundamentals. Businesses convert inputs (capital, materials, staff talent) into valuable outputs and generate valuable surplus (margin) for shareholders and other stakeholders.

This act of **conversion** is a **process**. The overall process has been described by Michael Porter as a "value chain."



Value Chain similar to [Porter1998].

Note the progression along the bottom: first, raw materials are acquired, then manufactured, then distributed at which point they are marketed and sold, and ultimately serviced. Porter termed this

set of activities "Primary."

The upper half represents what Porter termed "support." Supporting activities represent necessary, but not value adding, functions.

There are many criticisms of the Porter value chain. Without research and development, there is no product, so why is it considered a support activity (and confusingly named just "Technology Development")? Surely, marketing and sales help drive demand for the product, especially with manufacturing lead times shrinking, so why do they appear so late in the game?

Overall, the model represents a manufacturing-centric world where customers were happy to buy what was on offer.

However, this does not change the fact that most value has some rough ordering to it (it is difficult to sell and service a product before you invent and produce it), and the idea that some activities are primary while others are supporting also is useful.

The value chain as a whole is high level. We can think of a large company in these terms, but we cannot easily manage it this way. Value chains therefore are the ultimate "process" but in order for process to have value, we need to take it down a level or two.

In order to do this, we should clarify some terms.

Process and related terms

The following terms often appear in discussions of process:

- Function
- Capability
- Practice

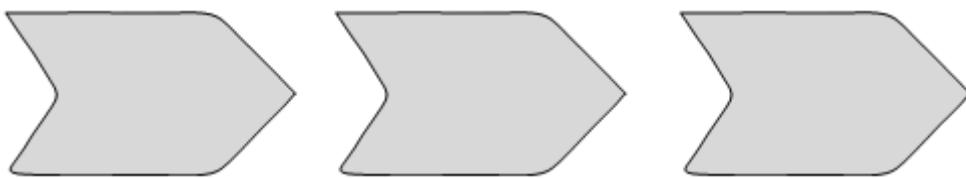
We will use the following definitions:

A **process** is an ordered, countable set of activities; an event-driven, value-adding sequence that can be measured and improved. The following terms often are used to describe processes or their components:

- Value chain
- Value stream
- Business process
- Workflow
- Procedure
- Task
- Step

Processes **cross** functions.

Visually, processes are often represented with chevrons, like Porter's original value chain:

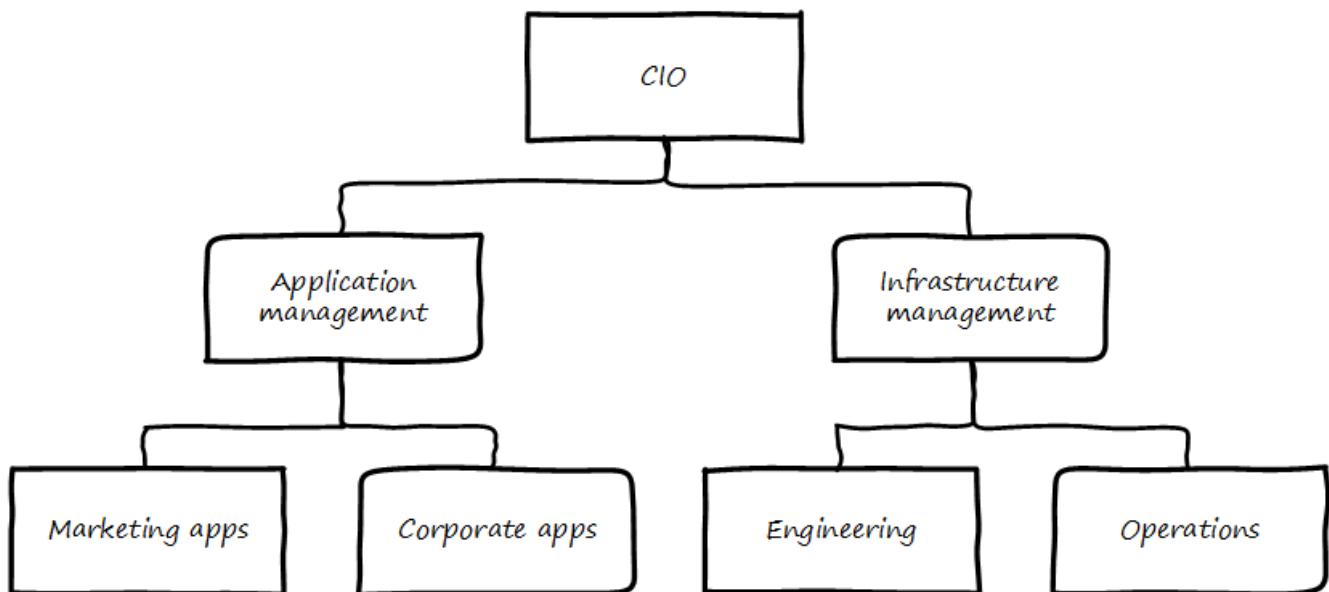


There is no industry standard that definitely lists the concepts above in terms of granularity. Is a "task" larger or smaller than a "procedure?" We will not discuss these much. However, between the high level concept of "value chain" and the concepts of "process" and "activity," is the concept of **value stream**. This is a key concept from the Lean community, and is a relatively higher-level concept (a value chain decomposes into value streams, according to some.)

An example of a value stream could be the complete manufacturing and sales process for one product family. Because they are more specific, value streams are more measurable and countable. Optimizing them in terms of eliminating wasted time, effort and inventory is an objective of Lean thinking. See *Learning to See* by Rother and Shook [[Rother2003](#)].

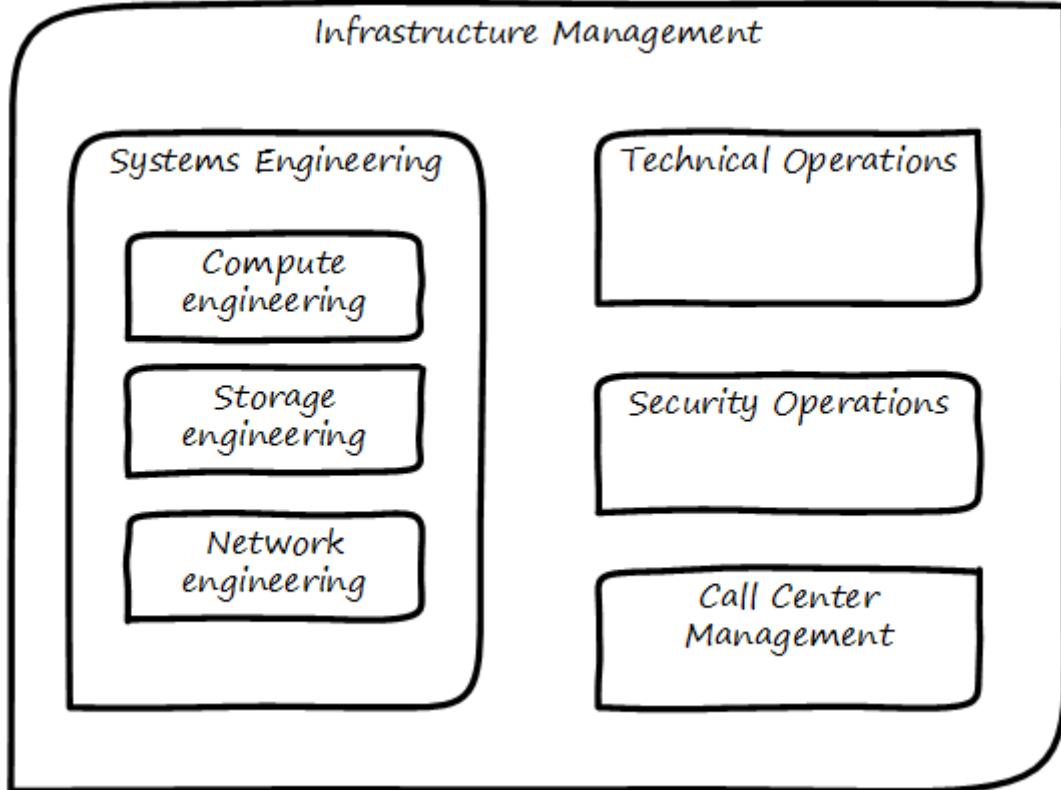
Functions are unordered, steady state domains of activity. An organizational chart often can be based on various functions. Functions also often imply hierarchy. They are more difficult to measure or improve; usually, improving a function means improving some process it owns or performs.

Functions, especially when they are closely aligned to an organizational structure, may be represented as a hierarchy:



They may also be represented as nested shapes:

Infrastructure Management

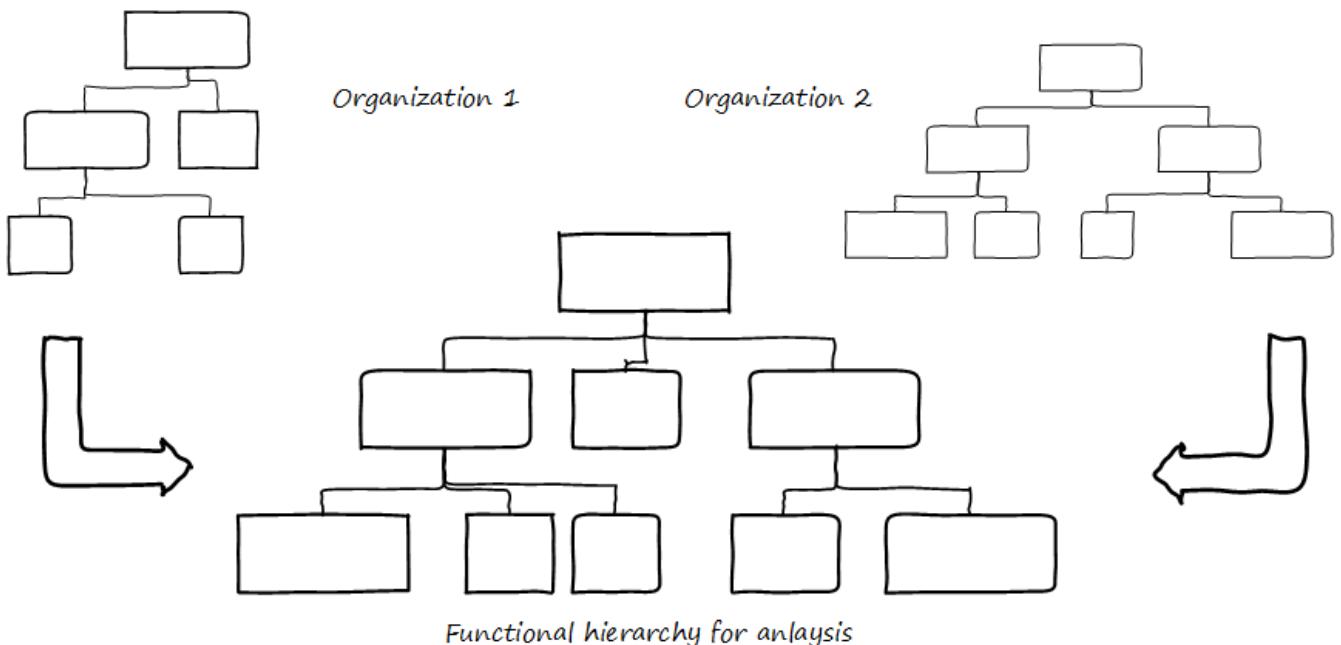


However - and this is an important point - functions do **not** necessarily need to reflect the organizational chart. Often, they are used as a "logical" construct, to help with organizational design.

What do we mean by this?

Suppose two companies merge. They may leave their organizational charts untouched for a time, because they do not know what parts of each company should merge together. They have two sets of developers, two sets of sales staff, and so on. They do not feel that either company's organizational structure is a good basis for the new, larger company.

To solve this, they might create a logical functional hierarchy, with no-one's name on any part. Where their actual organization structure (temporarily) has duplication, the logical hierarchy does not. They then start to populate it with individuals and teams from the previous companies and in this way develop a new organizational structure.



This kind of thinking can help the organization evolve over time. If different teams find themselves performing similar activities, a functional analysis might indicate a new organizational structure.

Finally, the following two terms are also unordered, but have different connotations from function:

- Capability
- Service (sometimes)

A **capability** is a generic term for a capacity or potential.

A **service** is a transactional act of value, simultaneously produced and delivered (more on this later).

Finally, the concept of **practice** is the least formalized of the concepts. It represents a loose, conceptual area that may represent a community of common interest (for example, a Java community of practice). It's not usually measured or tracked for improvement. Some topics may start as a practice and formalize into a function, for example Capacity management.

Review the section on the [Spotify model](#). How does it relate to the discussion here?

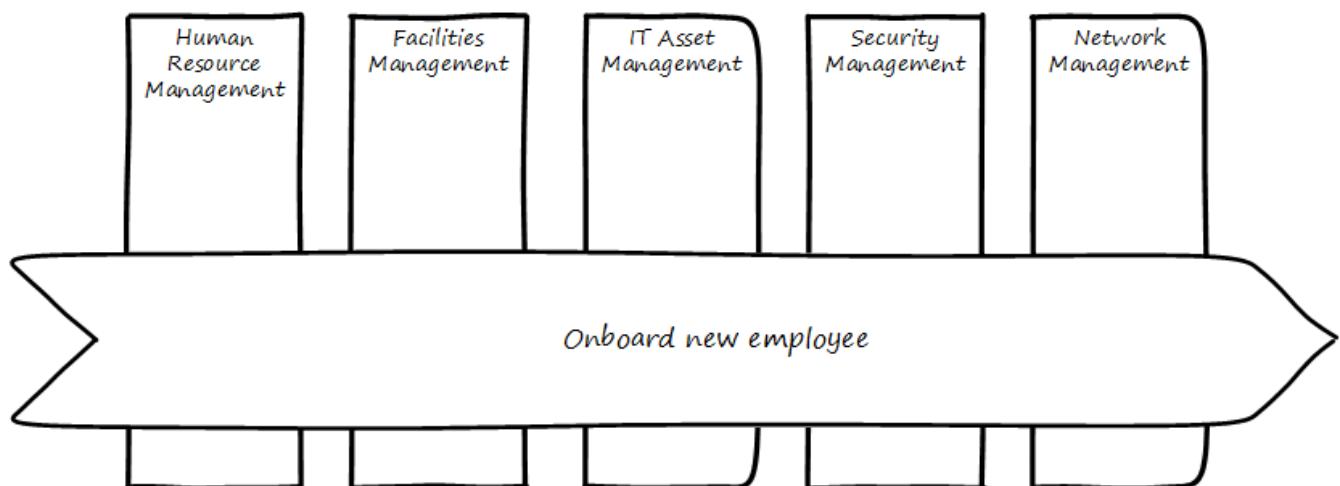
We say that an IT management capability may be implemented by a process, function, or practice.

A deeper look at processes and functions

Processes cross functions in a repeatable way. You may have deep organizational functions such as

- Human Resource Management
- Facilities Management
- IT Asset Management
- Security Management
- Network Management

However, you have a value adding activity of bringing in a new employee that involved ALL of these functions:



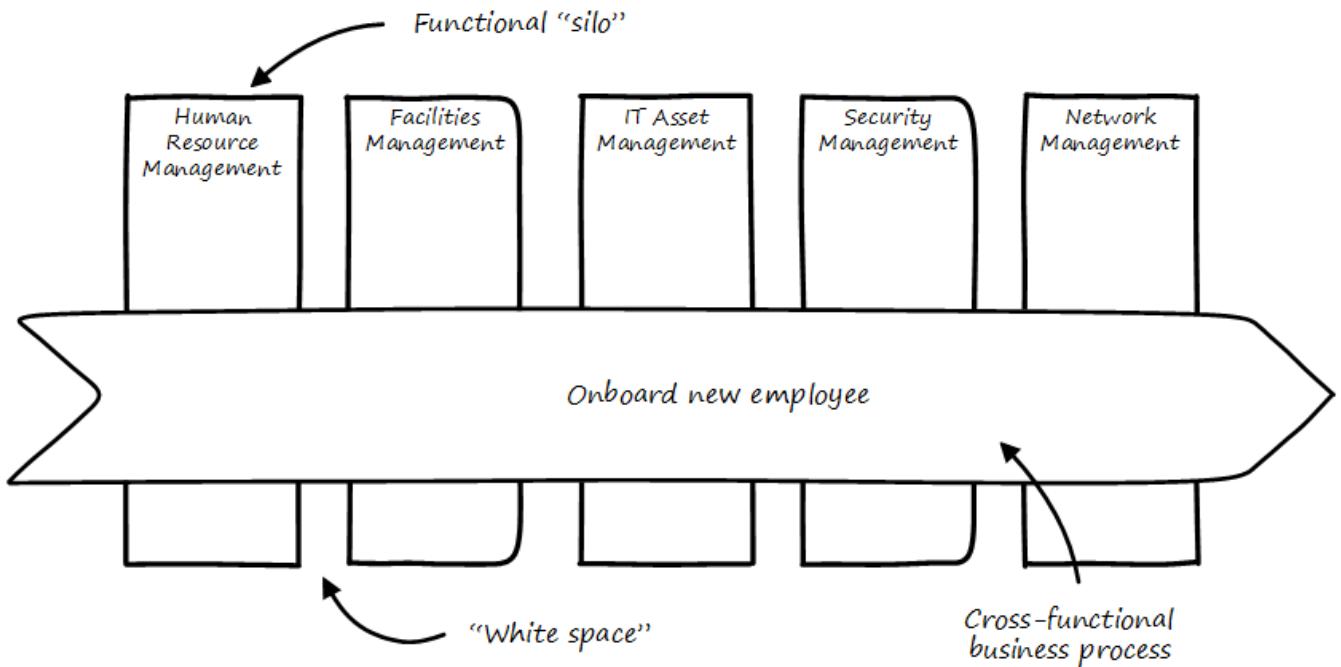
1. The HR department enters the employee in the HR system (including payroll).
2. Either the HR department or the hiring manager requests space from Facilities Management.
3. A workstation and smart phone are requested from IT Asset Management
4. When those devices are received, Security authorizes them (and the employee) to enterprise systems (critically the enterprise directory)
5. If a wired connection is needed, Network Management may be required to "turn on" a port in the employee's location.

This is a deliberately older, traditional workflow. Modern organizations may accelerate things:

1. HR enters a record
2. All employees use common "hoteling" space, so the major Facilities concern is whether the space can accommodate the peak # of employees seen (e.g. at 10 AM on a Wednesday)
3. Equipment is either BYOD (Bring Your Own Device) or is ordered from an authorized vendor and drop-shipped directly to the employee
4. In either case, security must establish some control over the devices, so that organizational data remains secure. There are various approaches for this, even for devices owned by the employee.
5. Networking in modern organizations is almost all wireless, but the employee still requires the appropriate credentials, likely provided by security. As with facilities, Networking's concerns now move to questions of capacity and availability of the overall infrastructure.

Notice however that even in the modern scenario, we have an end to end, repeatable value flow that crosses distinct domains of concern. The employee is not productive until all these steps are taken. Since the employee starts drawing payroll on hire, there is a substantial cost of delay for this entire process. A week's unproductive time could amount to \$10,000 or more, wasted.

Such delays can easily happen when functional areas are focused just on their responsibility, and no one is concerned for the overall process. This is why Rummler and Brache chose the title *Improving Performance: How to Manage the White Space on the Organization Chart* for their groundbreaking book on business process. The "white space" they are talking about is the spaces between the functions, also termed "silos" or "towers:"

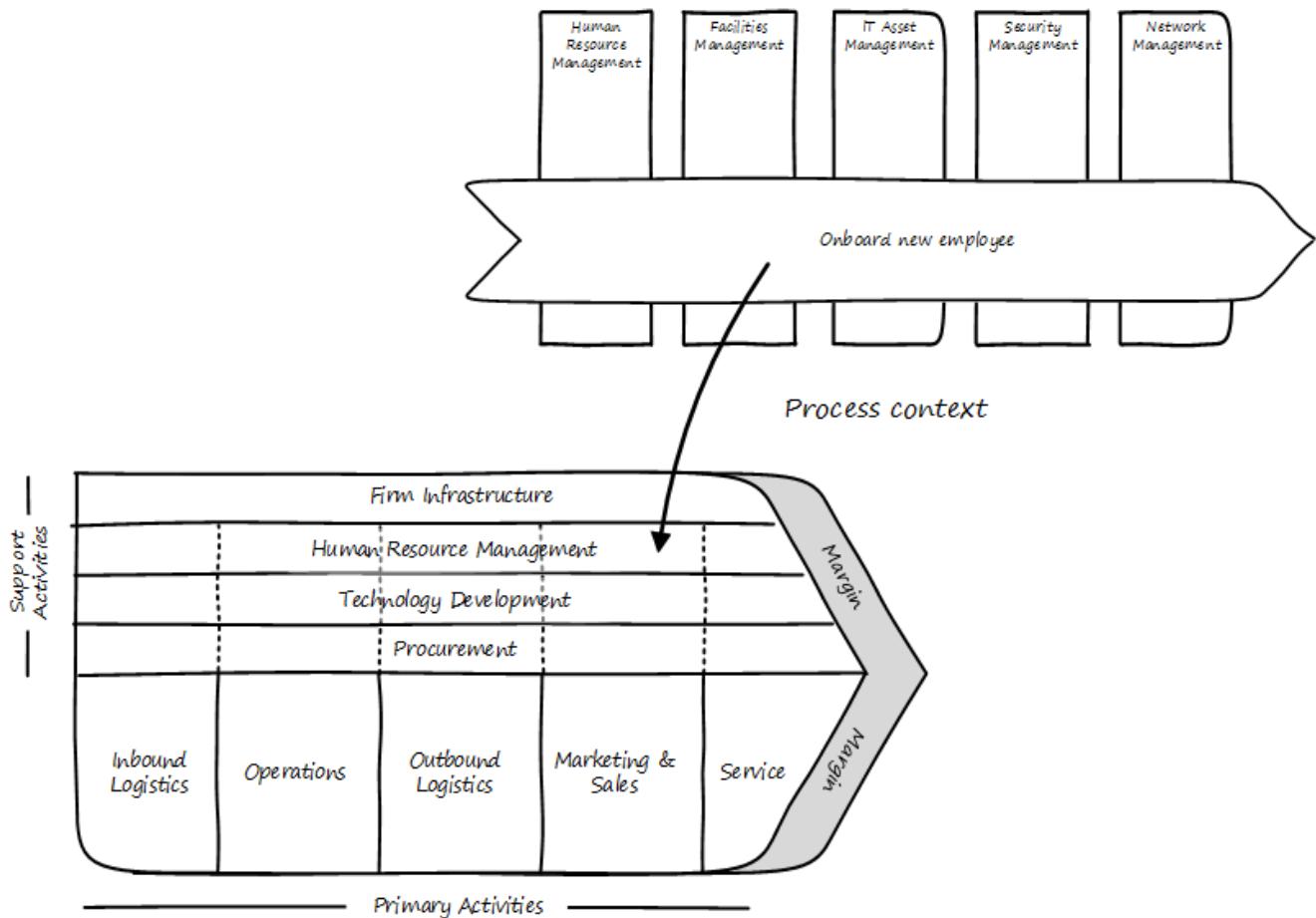


You may notice that the cross functional process is named with an active verb, "Onboard." This is in contrast to the functional silos, which end in "management." This is deliberate and a well-established BPM best practice. "Onboard employee" is event-driven and countable; it has a clear beginning and end and therefore can be measured in various ways. "Human Resource Management", on the other hand, has no beginning or end; it is a steady-state set of organizational concerns and resources.

IMPORTANT

The idea that processes are active and countable, while functions are passive and ongoing, is critical to effective organizational design and performance management.

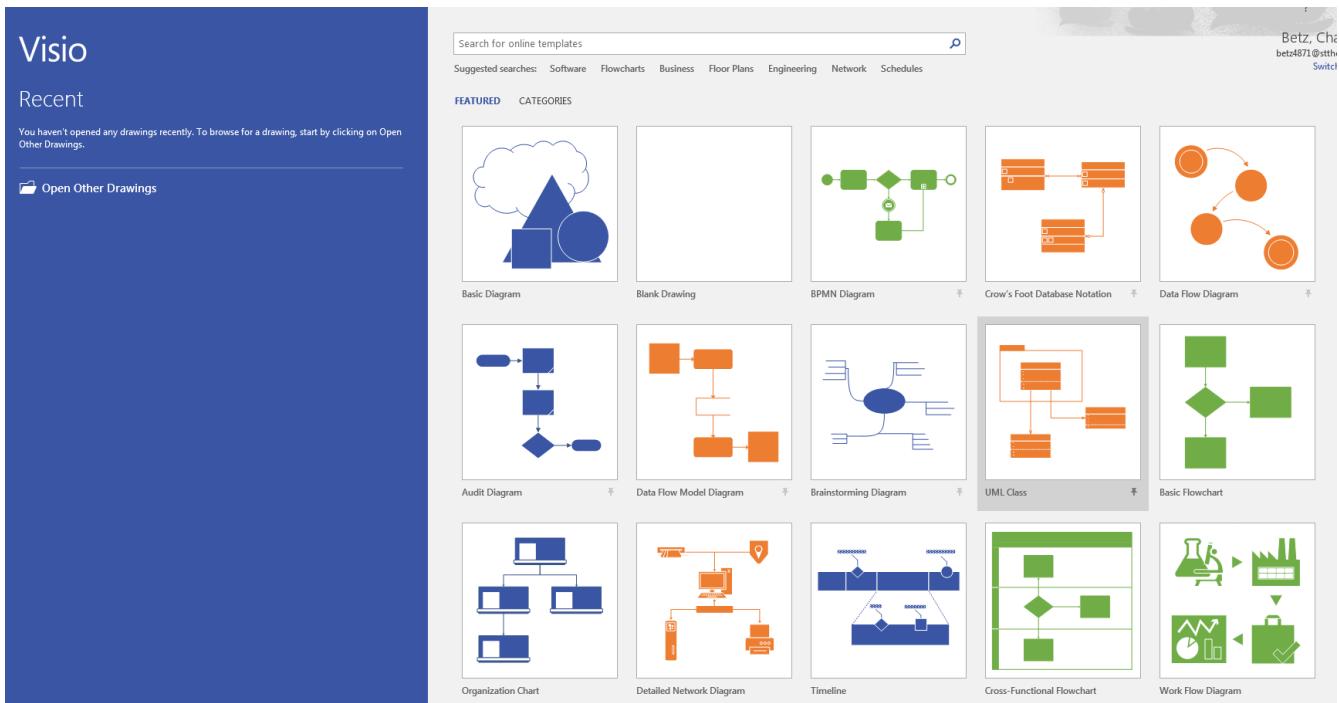
Finally, context is everything. Notice that this end to end process of "onboard employee" belongs to the overall supporting activity of Human Resource Management in terms of the value chain we started with:



There is a rich variety of ways to think about enterprises and their operations at the largest scale. We will return to the question of process context and value chain in the chapter section on process frameworks. For now, we will move to a more detailed level of process modeling, now that we have introduced certain key concepts.

Process modeling

If you page back through this book, you will see many graphics: some are pictures from real life, while others are conceptual. The practice of communicating concepts through graphical forms has a long history and great breadth of application. If you have access to the Microsoft product Visio, you can get a sense of this through looking at the various templates and solutions:



The idea that a box can represent an activity or objective, and can be linked in sequence to other boxes through lines, has been around since at least the 1920s. Many different visual languages have been developed over the years for various purposes. Some are highly formalized, others less so. The following definitions may help:

A **modeling language** is a set of concepts and definitions that describes a given problem space. Modeling languages can be formalized to different degrees of precision.

A **notation** is a specific set of visual symbols mapped onto a modeling language. One modeling language might have different notations; for example, Data Flow Diagrams have two notations but both communicate essentially the same language.

A **model** is a specific usage of a modeling language and notation to communicate some idea about a system. It may also be a verb, e.g. "I am going to **model** the system."

There are many different forms of modeling, extending far beyond IT and digital management. Scientists develop models of physical processes; financial analysts develop models of the stock market. Different aspects of a digital system might be modeled at different times:

- The process it is automating might be modeled as part of analysis
- Its data structures might be modeled somewhat later
- When the system is running under load, capacity analysts might model its consumption of compute and network resources and develop projections for when additional capacity might be needed

Process modeling is therefore only a subset, but an important one. Process analysis is one of most important activities in the broader area of business analysis and architecture, which in turn are key to **product management** in general (as we briefly mentioned in Chapter 4).

There are many process modeling languages and notations:

- Generic flowcharting
- The IDEF family
- The Unified Modeling Language
- Business Process Modeling Notation

There are many references on these in the concluding chapter section. We will only discuss some of the more important concepts in this section:

- The IGOE foundation (Input/Guide/Output/Enabler)
- Ordering, conditionality, and synchronization
- Swimlanes

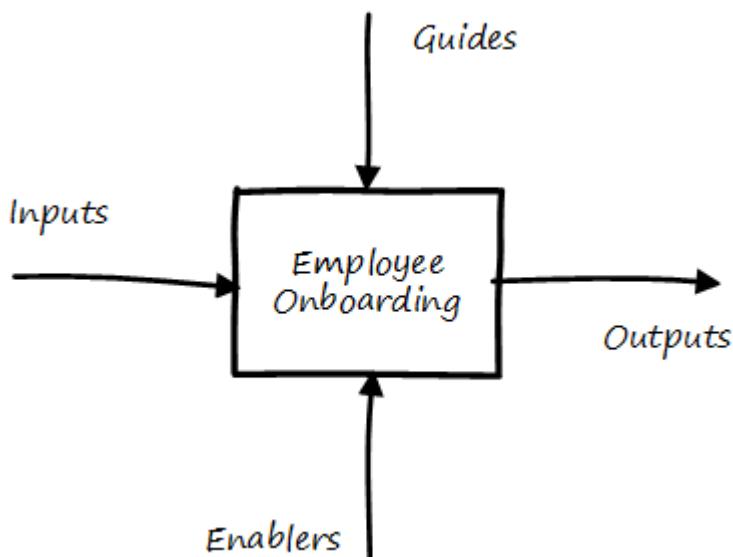
IGOE (Input/Guide/Output/Enabler)

A useful framework for thinking about activities represented as icons is the IGOE approach [BPTrends2013].

insert Process Renewal Group cite

IGOE stands for:

- Inputs
- Guides
- Outputs
- Enablers



Inputs include the information or event stimulus that initiates the activity. They also include resources that will be consumed in the course of the activity.

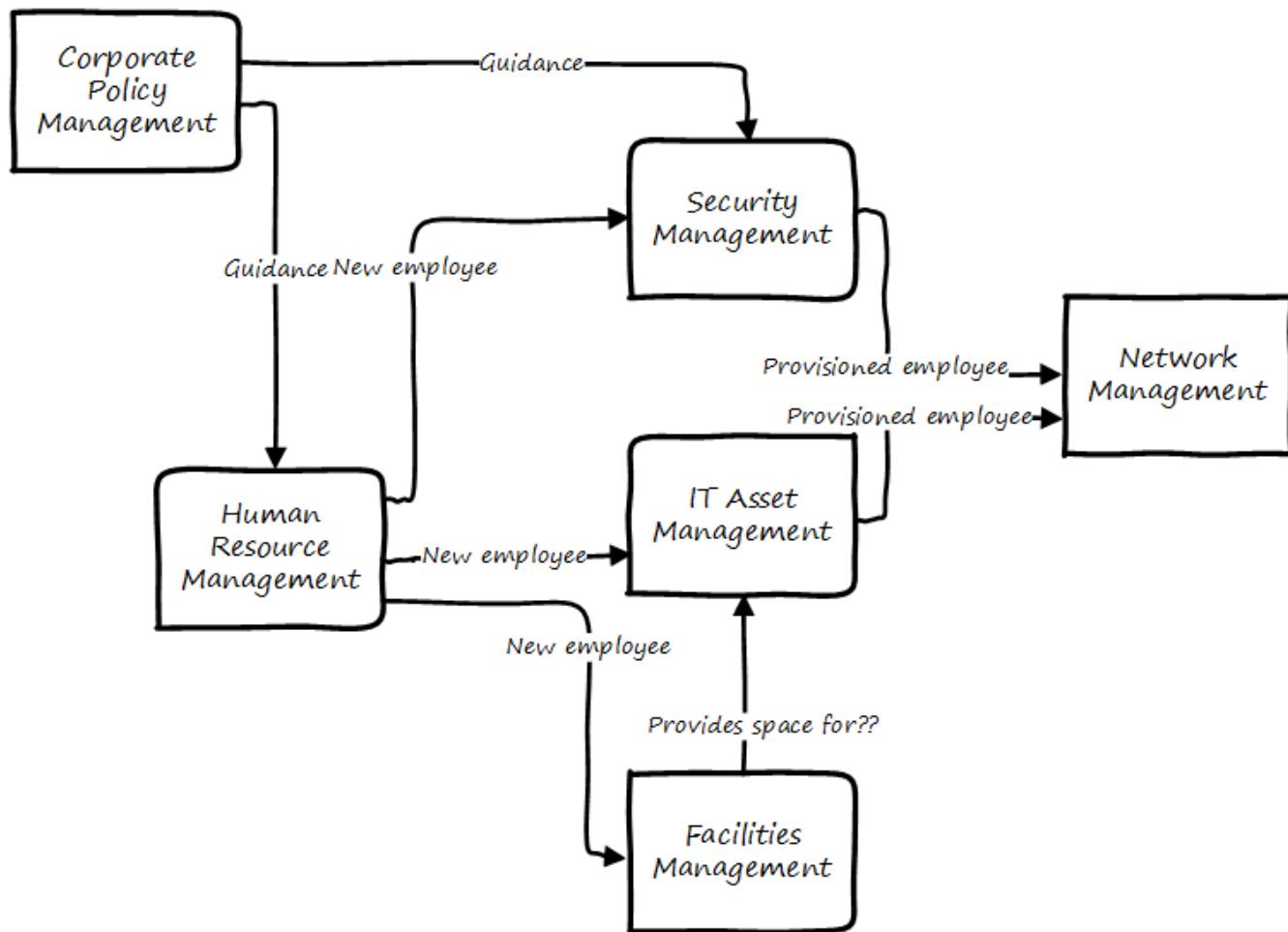
Guides govern the process, setting constraints and expectations (e.g. current HR policies). They are not consumed.

Outputs include the final, value-added outcome of the activity, and also serve as a signal for downstream activities to commence.

Enablers support the process, without being immediately consumed. Examples include infrastructure and facilities support.

The IGOE approach is seen in the classic IDEF0 standard (the terms used there were Inputs, Controls, Outputs, Mechanisms). This is a form of **functional modeling** which is **not** event driven but rather is used to scope higher level interactions between areas.

The original intent of the IDEF0 standard was for a multi-level, decomposition approach. A high level scope would be elaborated thus:



However, there is not a clear sense of the events, since the primary objects are still the functions. There also is the temptation to insert extraneous information, as indicated in the line between Facilities Management and IT Asset Management labeled "Provides space for ??".

It is true that Facilities Management does provide space for the IT Asset Management Team (in fact, all of these functions provide services to each other), but to draw this would result in a "spaghetti" diagram with several dozen lines and not much explanatory power.

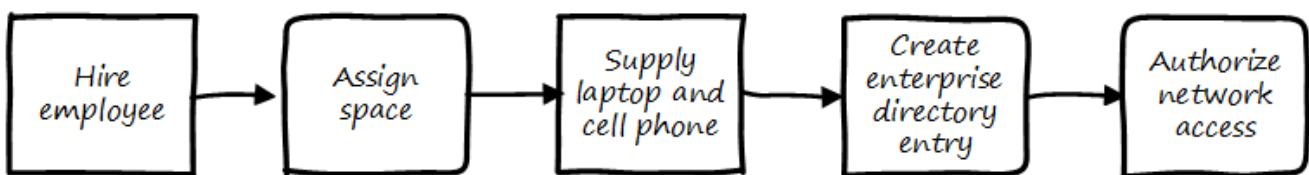
Purely functional modeling of this nature is therefore limited in usefulness, although it can be a good place to start.

Ordering, synchronization, and conditionality

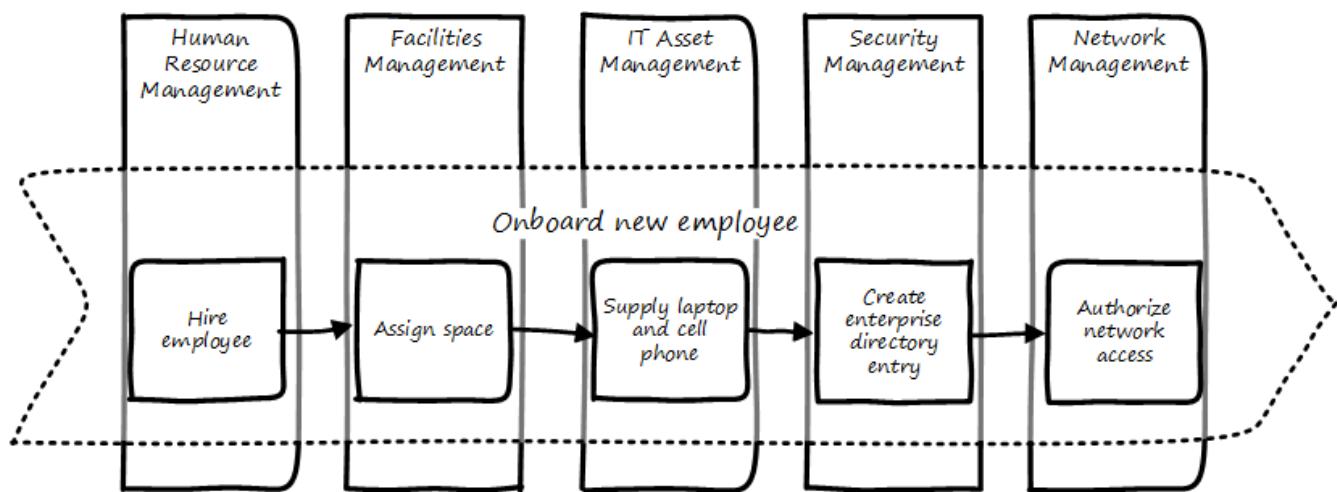
Activities in process modeling, by contrast, emphasize active verbs:

- Hire
- Assign
- Supply
- Create
- Authorize

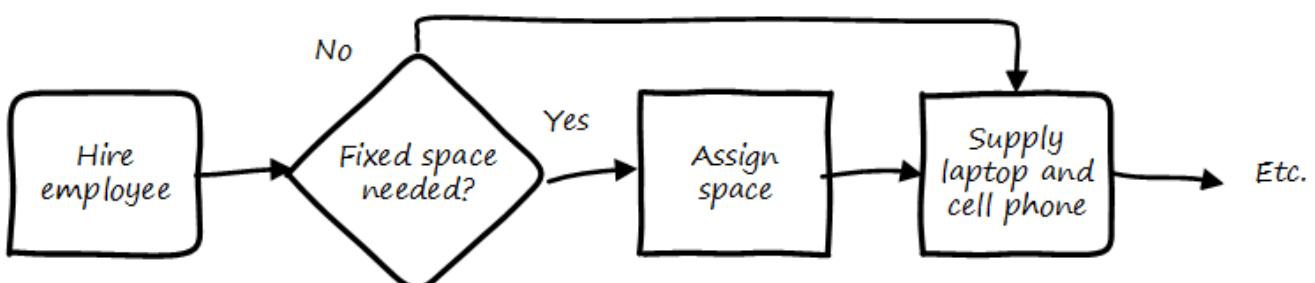
Multiple activities can be sequenced and this is often represented as a series of shapes connected by arrows:



Notice these activities align to the overall cross-functional flow we discussed above:

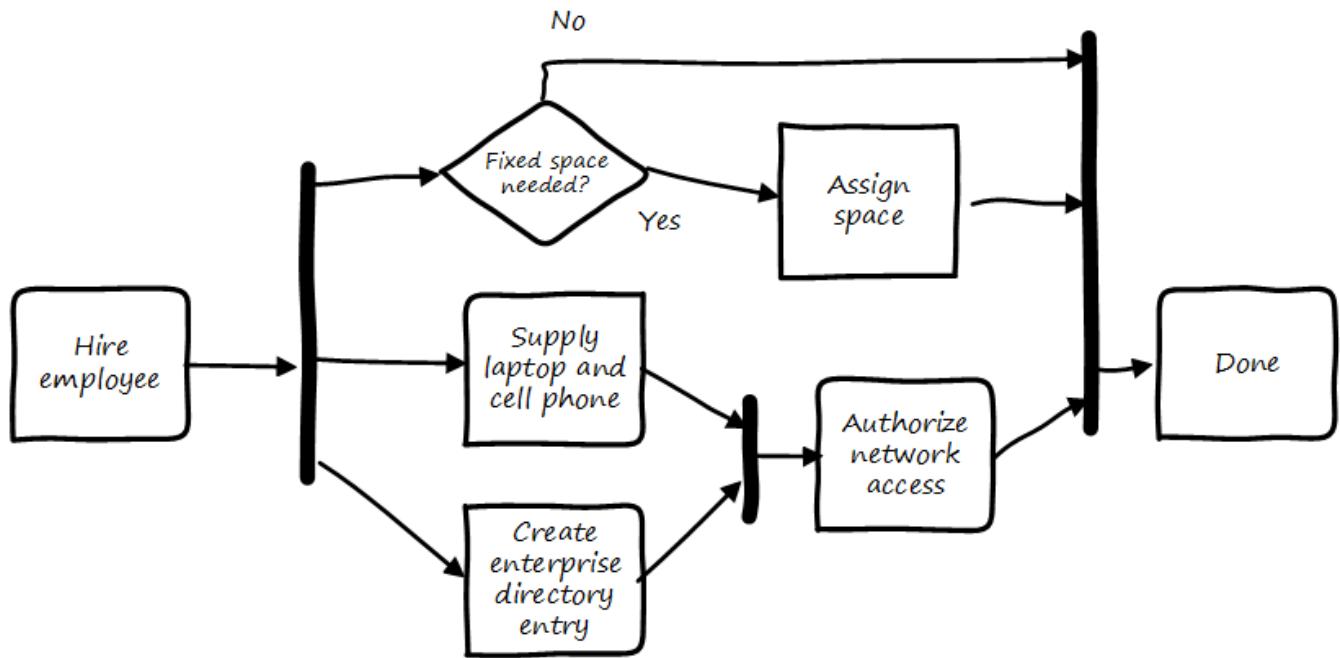


However, we encounter a complication. In order to describe this process accurately, we should note that some people do not have space assigned, if they are primarily telecommuting and/or working in "hoteling" space. We want to show a decision point, which in many notations is represented by a diamond.



Finally, this ordering is taking too long, because people are waiting for space to be assigned before doing anything else. This is a well understood problem; processes can branch and run in parallel (you may well be writing code that works this way; human processes also do this). In UML, the term

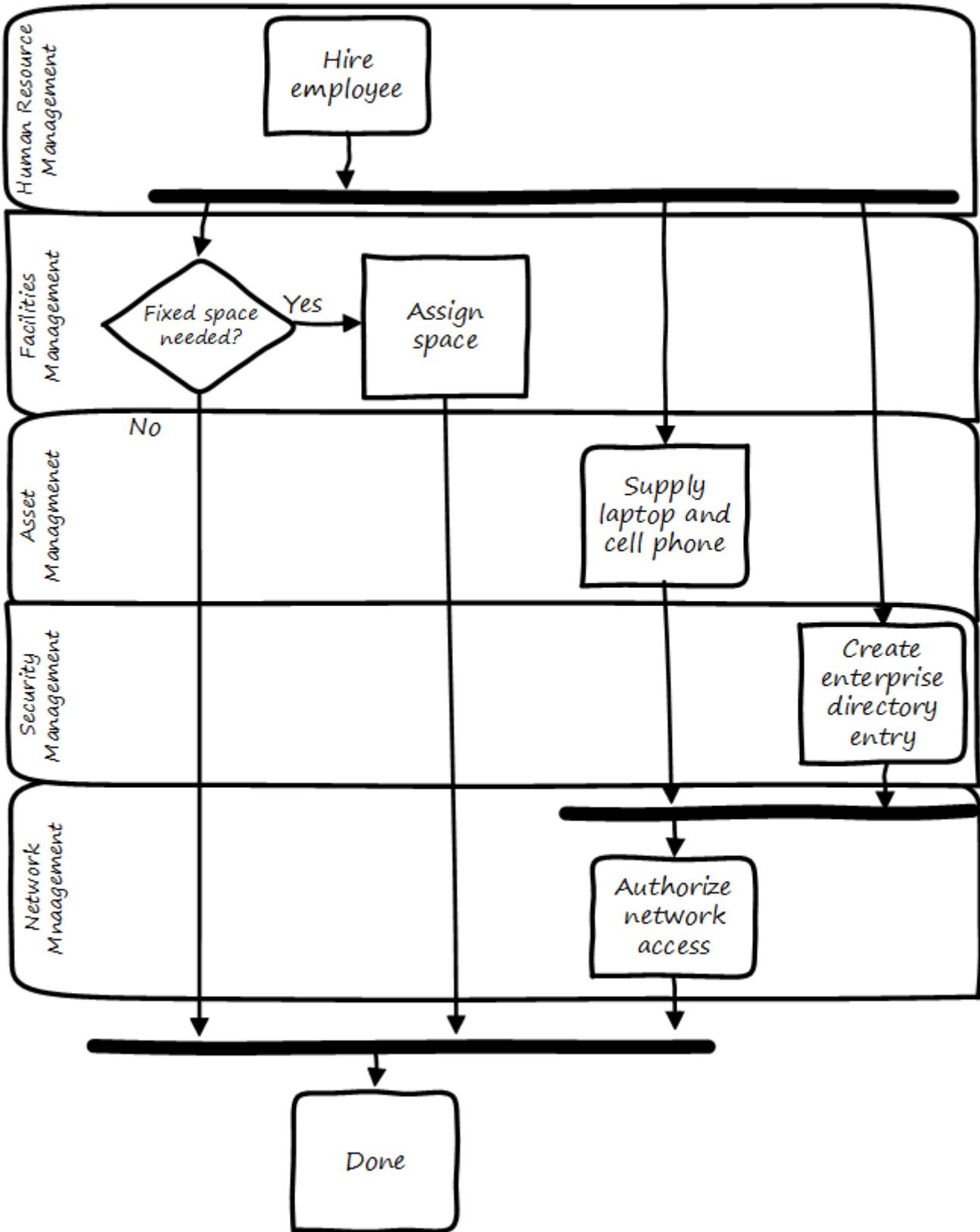
is "fork" for when processes diverge to run in parallel, and "join" when they must again synchronize for the next activity to continue:



In the above diagram, we see that 3 activities (including the decision point) can commence independently when the employee is hired, but network access requires that the employee have both an entry in the enterprise directory, as well as their hardware supplied. The entire process is complete when their space, equipment, and access needs have been met.

Swimlanes

We made much of the fact that processes cross functions, and yet with our final diagram we have lost our view of this fact. We can re-introduce the functional boundaries through the use of swimlanes (pioneered by Rummler in [\[Rummler1995\]](#)):



Adding the swimlanes helps us understand the cross-functional handoffs, and also where certain activities will remain within the scope of one function. Another interesting, finer point is the question of the sync points. Observe the sync point in the Network Management swimlane. Its presence implies that the Network Management function must confirm that the employee has equipment and a directory in the enterprise directory.

You may notice that this diagram has been "rotated." The diagram was becoming too long if drawn left to right. UML Activity Diagrams frequently use a top-to-bottom approach. The meaning of the

diagram is not changed.

A final caution on technique

The techniques briefly discussed here can be applied and combined in many different ways, and result in models of large scope and complexity. However there are crucial limitations to visual representations of complexity.

There are endless nuances and debates about BPM notations, methodology, and philosophy. Don't get caught up in it. Modeling should generate simple, clear artifacts that are useful by a broad audience, and it should not become an end in itself, or (in general) consume inordinate amounts of time. We will discuss this further in the chapter section on Agile.

With the above discussion, we have covered in a summary form the basics of business process modeling. In the next section on Agile and process management, we will discuss various process pitfalls and appropriate Agile and Lean responses.

Process control and continuous improvement

Collaborative

Status: This section is in first draft as of 8 Jun 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

a process can still be controlled even if it can't be defined.

— Martin Fowler, preface to Agile Software Development with Scrum

In this chapter section, we take a deeper look at the concept of process and how processes are managed and controlled. In particular, we will explore the concept of continuous (or continual) improvement and its rich history and complex relationship to Agile.

You are now at a stage in your company's evolution, or your career, where an understanding of continuous improvement is helpful. Without this, you will increasingly find you don't understand the language and motivations of leaders in your organization, especially those with business degrees or background.

NOTE

There is a debate over whether to use the term "continuous" or "continual" improvement. We will use "continuous" here as it is the more commonly seen. Advocates of "continual" argue it is the more grammatically correct.

The scope of the word "process" is immense. Examples include:

- The end to end flow of chemicals through a refinery
- The set of activities across a manufacturing assembly line, resulting in a product for sale
- The steps expected of a customer service representative in handling an inquiry
- The steps followed in troubleshooting a software-based system
- The general steps followed in creating and executing a project
- The overall flow of work in software development, from idea to operation

This breadth of usage requires us to be specific in any discussion of the word "process." In particular, we need to be careful in understanding the concepts of efficiency, variation, and effectiveness. These concepts lie at the heart of understanding process control and improvement and how to correctly apply it in the digital economy.

Companies institute processes because it has been long understood that repetitive activities can be optimized when they are better understood, and if they are optimized, they are more likely to be economical and even profitable.

We have emphasized throughout this book that the process by which complex systems are created is not repetitive. Such creation is a process of **product development**, not **production**. And yet, the entire digital organization covers a broad spectrum of process possibilities, from the repetitive to the unique. You need to be able to identify what kind of process you are dealing with, and to choose the right techniques to manage it.

(For example, the employee provisioning process flow shown in the previous chapter is simple and prescriptive. Measuring its efficiency and variability would be possible, and perhaps useful.)

There are many aspects of continuous improvement we won't cover in this brief section. Some of them (systems thinking, culture, and others) have already been covered in this book. This book is based in part on Lean and Agile premises, and continuous improvement is one of the major influences on Lean and Agile, so in some ways we come full circle.

We are focusing on continuous improvement in the context of processes and process improvement. We'll therefore scope this to a few concerns: efficiency, variation, effectiveness, and process control.

History of continuous improvement

NOTE

History is important. You may think your career is far removed from the early days of the industrial revolution, but the influence of early management thinkers such as Frederick Taylor remains. You need to be able to recognize when his ideas are being applied, especially if they are being applied inappropriately (as can easily happen in modern digital organizations).

The history of continuous improvement is intertwined with the history of 20th century business itself. Before the industrial revolution, goods and services were produced primarily by local farmers, artisans, and merchants. Techniques were jealously guarded, not shared. A given blacksmith might have two or three workers, who might all forge a pan or a sword in a different way. The term "productivity" itself was unknown.

Then the Industrial Revolution happened.

As steam and electric power increased the productivity of industry, requiring greater sums of capital to fund, a search for improvements began. Blacksmith shops (and other craft producers such as grain millers and weavers) began to consolidate into larger organizations, and technology became more complex and dangerous. It started to become clear that allowing each worker to perform the work as they preferred was not feasible.

Enter the scientific method. Thinkers such as Frederick Taylor and Frank and Lillian Gilbreth (of "Cheaper by the Dozen" fame) started applying careful techniques of measurement and comparison, in search of the "one best way" to dig ditches or assemble vehicles. An entire profession of industrial engineering was established, along with the formal study of business management itself.

Frederick Taylor and efficiency

Frederick Taylor (1856-1915) was a mechanical engineer and one of the first industrial engineers. In 1911, he wrote *Principles of Scientific Management*.

One of Taylor's primary contributions to management thinking was a systematic approach to efficiency. To understand this, let's consider some fundamentals.

Human beings engage in repetitive activities. These activities consume inputs and produce outputs. It is often possible to compare the outputs against the inputs, numerically, and understand how "productive" the process is.

For example, suppose you have two factories producing identical kitchen utensils (pizza cutters, say).

If one factory can produce 50,000 pizza cutters for \$2,000, while the other requires \$5,000, the first factory is more productive.

Assume for a moment that the workers are all earning the same across each factory. There is possibly a "process" problem. The first factory is more *efficient* than the second; it can produce more given the same set of inputs. Why?

There are many possible reasons. Perhaps the second factory is poorly laid out and the work in progress must be moved too many times in order for workers to perform their tasks. Perhaps the workers are using tools that require more manual steps. Understanding the differences between the two factories, and recommending the "best way," is what Taylor pioneered, and what industrial engineers do to this day.

As Peter Drucker, one of the most influential management thinkers, says of Frederick Taylor:

The application of knowledge to work explosively increased productivity. For hundreds of years there had been no increase in the ability of workers to turn out goods or to move goods . . . But within a few years after Taylor began to apply knowledge to work, productivity began to rise at a rate of 3.5 to 4 percent compound a year—which means doubling every eighteen years or so. Since Taylor began, productivity has increased some fiftyfold in all advanced countries. On this unprecedented expansion rest all the increases in both standard of living and quality of life in the developed countries. [Drucker1993], pp.37-38.

W.E. Deming and variation

The quest for efficiency leads to the long-standing management interest in variability and variation. What do we mean by this?

If you expect a process to take 5 days, what do you make of occurrences when it takes 7 days? 4 days? If you expect a manufacturing process to yield 98% usable product, what do you do when it falls to 97%? 92%?

In highly repeatable manufacturing processes, statistical techniques can be applied. Analyzing such "variation" has been a part of management for decades, and is an important part of disciplines such as Six Sigma. This is why Six Sigma is of such interest to manufacturing firms.

W. Edwards Deming (1900-1993) is noted for (among many other things) his understanding of variation and organizational responses to it. Understanding variation is one of the major parts of his "System of Profound Knowledge." He emphasizes the need to distinguish special causes from common causes of variation; special causes are those requiring management attention.

Deming in particular was an advocate of the control chart, a technique to understand whether a process was within statistical control:

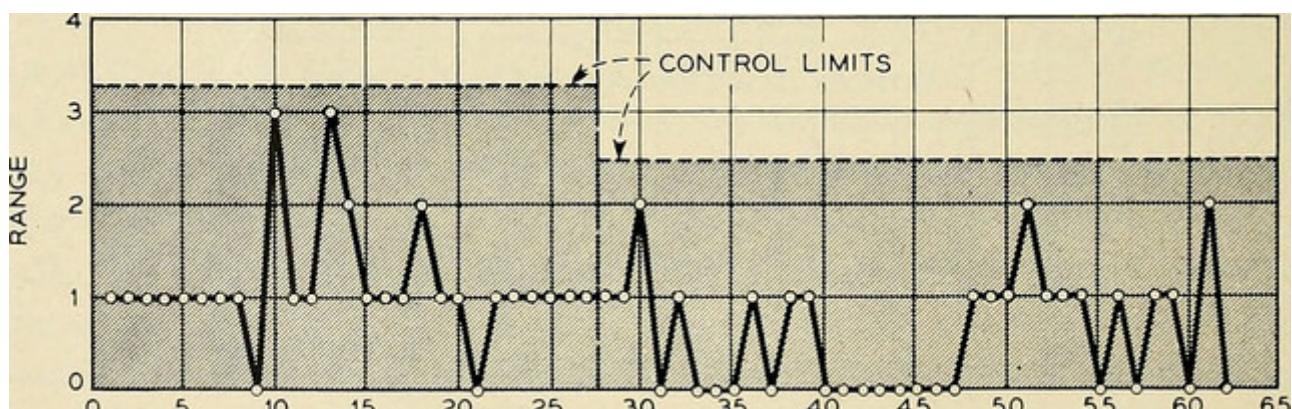


image from <https://www.flickr.com/photos/internetarchivebookimages/14775750673/>, no known copyright restrictions

However, using techniques of this nature makes certain critical assumptions about the nature of the process. Understanding variation and when to manage it requires care. These techniques were defined to understand **physical** processes that in general follow [normal distributions](#).

IMPORTANT

Learning some statistics is essential, if you want to progress in your career.

This section assumes you are comfortable with the concept of a "distribution" and in particular what the "normal distribution" is.

For example, let's say you are working at a large manufacturer, in their IT organization, and you see the a metric of "variance from project plan." The idea is that your actual project time, scope and resources should be the same, or close to, what you planned. In practice, this tends to become a discussion about time, as resources and scope are often fixed.

The assumption is that, for your project tasks, you should be able to estimate to a meaningful degree of accuracy. Your estimates are equally likely to be too low, or too high. Furthermore, it should be somehow possible to improve the accuracy of your estimates. Your annual review depends on this, in fact.

The problem is that neither of these are true. Despite heroic efforts, you cannot improve your estimation. In process control jargon, there are too many causes of variation for "best practices" to emerge. Project tasks remain unpredictable, and the variability does not follow a normal distribution. Very few tasks get finished earlier than you estimated, and there is a [long tail](#) to the right, of tasks that take 2x, 3x or 10x longer than estimated.

In general, applying statistical process control to variable, creative product development processes is inappropriate. For software development, Steven Kan states: "Many assumptions that underlie control charts are not being met in software data. Perhaps the most critical one is that data variation is from homogeneous sources of variation." That is, the causes of variation are knowable and can be addressed. This is in general not true of development work. [\[Kan2003\]](#)

Deming (along with Juran) is also known for "continuous improvement" as a cycle, e.g. "Plan/Do/Check/Act." Such cycles are akin to the scientific method, as they essentially engage in the ongoing development and testing of hypotheses, and the implementation of validated learning. We have touched on similar cycles in our discussions of [Lean Startup](#) and [Toyota Kata](#).

Lean Product Development and cost of delay

the purpose of controlling the process must be to influence economic outcomes. There is no other reason to be interested in process control.

— Don Reinertsen, Managing the Design Factory

Discussions of efficiency usually focus on productivity given a certain set of inputs. Time can be one of those inputs. Everything else being equal, a company that can produce the pizza cutters more quickly is also viewed as more efficient. Customers may pay a premium for early delivery, and may penalize late delivery; such charges typically would be some percentage (say plus or minus 20%) of the final price of the finished goods.

However, the question of time becomes a game-changer in the "process" of new product development. As we have discussed previously: Starting with a series of influential articles starting in the early 1980s, Don Reinertsen developed the idea of "cost of delay" for product development ([\[Reinertsen1997\]](#)).

Where the cost of a delayed product shipment might be some percentage, the cost of delay for a delayed product could be much more substantial. For example, if a new product launch misses a key trade show where competitors will be presenting similar innovations, the cost to the company might be millions of dollars of lost revenue - many times the product development investment. (Perhaps even orders of magnitude larger.)

This is not really a question of "efficiency;" of comparing inputs to outputs and looking for a few percentage points improvement. It is more a matter of effectiveness; of the company's ability to execute on complex knowledge work.

Scrum and empirical process control

process theory experts . . . were amazed and appalled that my industry, systems development, was trying to do its work using a completely inappropriate process control model.

— Ken Schwaber, Agile Software Development with Scrum

Ken Schwaber, inventor of the Scrum methodology (along with Jeff Sutherland), like many other software engineers in the 1990s, experienced discomfort with the process control approach at the time. Mainstream software development processes sought to make software development predictable and repeatable in the sense of a defined process.

As Schwaber discusses ([\[Schwaber2002\]](#), pp. 24-25), defined processes are completely understood, which is not the case with creative processes. Highly automated industrial processes run predictably, with consistent results. By contrast, complex processes that are not understood require the **empirical model**.

NOTE The defined and empirical models of process control are concepts from formal [process control theory](#). We touched on some related topics in our discussion of systems theory and [feedback](#).

Empirical process control relies on frequent inspection and adaptation. After exposure to Dupont process theory experts who clarified the difference between defined and empirical process control, Schwaber went on to develop the influential Scrum methodology. As he notes:

During my visit to DuPont . . . I realized why [software development] was in such trouble and had such a poor reputation. We were wasting our time trying to control our work by thinking we had an assembly line when the only proper control was frequent and first-hand inspection, followed by immediate adjustments. [\[Schwaber2002\]](#), p. 25.

Metrics, measurements, and KPIs

needs work

We've deferred discussion of metrics and KPIs until now, due to the risk of their mis-application.

Operations (often in the form of business processes) generate data, and data can be aggregated and reported on. Examples of metrics might include:

- Quarterly sales as a dollar amount
- Percentage of time a service or system is available
- Number of successful releases or pushes of code (new functionality)

Such reporting serves as a form of feedback for management, and even governance. A variety of terms and practices exist, such as:

- The Balanced Scorecard
- The concept of a metrics hierarchy
- Leading versus lagging indicators

Balanced Scorecard

The balanced score card is a commonly-seen approach for measuring and managing organizations. First proposed by Kaplan and Norton [\[Kaplan1992\]](#) in the Harvard Business Review, the Balanced Scorecard groups metrics into the following subject areas:

- Financial
- Customer
- Internal business processes
- Learning and growth

Metrics Hierarchy

Metrics can be seen as "lower" versus "higher" level. For example, the metrics from a particular product might be aggregated with the metrics from all products, to provide an overall metric of product success.

Leading & Lagging Indicators

Metrics can indicate past performance (lagging), or predict future performance (leading).

Process frameworks

Collaborative

Status: This section is in first draft as of 9 Jun 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

We will now turn to a critical examination of the IT management frameworks. While there is much of value in them, there are many places where they may lead you into the "defined process" trap, and they may not provide enough support for the alternative approach of empirical process control.

Defining frameworks

NOTE

There are other usages of the term "framework," especially in terms of software frameworks. Process and management frameworks are non-technical.

So, what is a "framework?"

The term "framework," in the context of business process, is used for comprehensive and systematic representations of a major business area's activities. In general, an industry framework is a structured artifact that seeks to articulate a professional consensus regarding a domain of practice. The intent is usually that the guidance be mutually exclusive and collectively exhaustive within the domain, so that persons knowledgeable in the framework have a broad understanding of domain concerns.

The first goal of any framework, for a given conceptual space, is to provide a "map" of its components and their relationships. Doing this serves a variety of goals:

- Develop and support professional consensus on the business area
- Support training and orientation of professionals new to the area (or its finer points)
- Support governance and control activities related to the area (more on this in Chapter 10)

An overview of the major frameworks

Many frameworks have emerged in the IT space, with broader and narrower domains of concern. Some are owned by non-profit standards bodies; others are commercial. We will focus on five in this book. In roughly chronological order, they are:

- CMMI (Capability Maturity Model-Integrated)

- ITIL (originally the Information Technology Infrastructure Library)
- PMBOK (The Project Management Body of Knowledge)
- COBIT (aka Control Objectives for Information Technology)
- TOGAF (The Open Group Architecture Framework)

A 6th framework, IT4IT, will be discussed in Chapter 12.

CMMI

The Capability Maturity Model was originally published by Watts Humphrey in 1989 in the book *Managing the Software Process* [[Humphrey1989](#)].

The CMM was later taken on by the Software Engineering Institute at Carnegie-Mellon University, and is now managed by the CMMI Institute, which was acquired in 2016 by ISACA.

The CMM has had considerable influence on software development and IT management more generally. It popularized the idea that process management tends to evolve in the same way across organizations, following a "staged" model:

Stage 1: Initial

Stage 2: Repeatable

Stage 3: Defined

Stage 4: Managed

Stage 5: Optimizing

(These are Humphrey's original levels. There are some differences with the current CMMI, but the concept is the same.)

These ideas can be seen reflected in later guidance such as COBIT and ITIL.

Within this overall framework of maturation, CMMI currently calls for 22 process areas:

- Causal Analysis and Resolution (CAR)
- Configuration Management (CM)
- Decision Analysis and Resolution (DAR)
- Integrated Project Management (IPM)
- Measurement and Analysis (MA)
- Organizational Process Definition (OPD)
- Organizational Process Focus (OPF)
- Organizational Performance Management (OPM)
- Organizational Process Performance (OPP)

- Organizational Training (OT)
- Product Integration (PI)
- Project Monitoring and Control (PMC)
- Project Planning (PP)
- Process and Product Quality Assurance (PPQA)
- Quantitative Project Management (QPM)
- Requirements Development (RD)
- Requirements Management (REQM)
- Risk Management (RSKM)
- Supplier Agreement Management (SAM)
- Technical Solution (TS)
- Validation (VAL)
- Verification (VER)

CMMI for a time was a critical requirement for obtaining US Government contracts, especially from the Department of Defense. As a result, outsourcing firms devoted significant effort and resources to becoming "Level 5" certified.

ITIL

About the same time as the CMM's initial creation and transition to the Software Engineering Institute, the Central Computer and Telecommunications Agency of the United Kingdom recognized that IT practices were becoming fragmented. In response, a multi-volume set of guidance known as the IT Infrastructure Library was created. A 2nd edition consolidated and updated this guidance, and achieved worldwide acceptance as a de facto standard for IT management. The term "service" was inserted, i.e. "IT Service Management" but the distinction between the ITSM term and simple "IT Management" has always been somewhat unclear.

Due to the particular structure of UK government publications, ITIL has never covered project management or the SDLC per se, or analysis, architecture and design.

ITIL is currently published via the Axelos commercial joint venture between the UK government and Capita Group. It is now in its 4th major edition (2011). While often considered an "operational" framework, ITIL spans the lifecycle of IT services and systems; the current volumes reflect a lifecycle:

- Service Strategy
- Service Design
- Service Transition
- Service Operations
- Continual Service Improvement

Within these volumes, ITIL defines a number of activities, functions, and what it calls processes (more on this below). These definitions have helped stabilize industry practice and provided a basis for industry training and certification of individuals.

ITIL's processes include:

ITIL Stage	Processes
Service Strategy	Strategy Management for IT Services Service Portfolio Management Demand Management Financial Management for IT Services Business Relationships Management
Service Design	Design Coordination Service Catalogue Management Service Level Management Risk Management Capacity Management Availability Management IT Service Continuity Management Information Security Management Compliance Management Architecture Management Supplier Management
Service Transition	Change Management Change Evaluation Project Management (Transition Planning and Support) Application Development Release and Deployment Management Service Validation and Testing Service Asset and Configuration Management Knowledge Management
Service Operation	Event Management Incident Management Request Fulfillment Access Management Problem Management IT Operations Control Facilities Management Application Management Technical Management
Continual Service Improvement	Service Review Process Evaluation Definition of CSI Initiatives Monitoring of CSI Initiatives

PMBOK

The Project Management Body of Knowledge is a publication of the Project Management Institute. It represents the codification of formal project management knowledge. There is a comparable Axelos publication, Prince2, not covered here. PMBOK was covered in [Chapter 8](#).

COBIT

COBIT (originally the Control Objectives for Information Technology) is a set of guidance from ISACA (originally the IS Audit and Control Association). It has a broader scope than ITIL, as it includes architecture and project management. Where ITIL contains lengthy and detailed narrative, COBIT is more terse and structured.

We will discuss the concepts of IT governance and controls in the next section, and therefore at this point in the reading are not completely ready to evaluate COBIT in terms of its initial objectives. (In COBIT terms, processes are just one of various "enablers" that can serve as controls for enterprise risk.) However, COBIT is widely used as a reference for understanding IT organizational processes and activities, and is discussed in that sense in this chapter.

The following processes are suggested by COBIT for IT management and governance. (*Governance*, the "EDM" processes, is very clearly distinguished from *management* in COBIT. We will discuss this in Chapter 10.)

As COBIT notes, "The proposed process model is a complete, comprehensive model, but it is not the only possible process model. Each enterprise must define its own process set, taking into account its specific situation." [\[ISACA2012a\]](#), p.32.

COBIT is strongly supportive of the standard CMMI/ISO/IEC 15504 process maturity progression and therefore is subject to the previous criticisms regarding the suitability of this approach for digital management, especially research and development processes and other less repeatable activities.

COBIT Domain	Process
Evaluate, Direct and Monitor (EDM) [Governance processes]	EDM01 Ensure Governance Framework Setting and Maintenance EDM02 Ensure Benefits Delivery EDM03 Ensure Risk Optimisation EDM04 Ensure Resource Optimisation EDM05 Ensure Stakeholder Transparency
Align, Plan and Organize (APO)	APO01 Manage the IT Management Framework APO02 Manage Strategy APO03 Manage Enterprise Architecture APO04 Manage Innovation APO05 Manage Portfolio APO06 Manage Budget and Costs APO07 Manage Human Relations APO08 Manage Relationships APO09 Manage Service Agreements APO10 Manage Suppliers APO11 Manage Quality APO12 Manage Risk APO13 Manage Security

COBIT Domain	Process
Build, Acquire and Implement (BAI)	BAI01 Manage Programs and Projects BAI02 Manage Requirements Definition BAI03 Manage Solutions Identification and Build BAI04 Manage Availability and Capacity BAI05 Manage Organisational Change Enablement BAI06 Manage Changes BAI07 Manage Changes Acceptance and Transitioning BAI08 Manage Knowledge BAI09 Manage Assets BAI10 Manage Configuration
Deliver, Service and Support (DSS)	DSS01 Manage Operations DSS02 Manage Service Requests and Incidents DSS03 Manage Problems DSS04 Manage Continuity DSS05 Manage Security Services DSS06 Manage Business Process Controls
Monitor, Evaluate and Assess (MEA)	MEA01 Monitor, Evaluate and Assess Performance and Conformance MEA02 Monitor, Evaluate and Assess the System of Internal Control MEA03 Evaluate and Assess Compliance with External Requirements

Each process is further elaborated into practices. For example, the process APO08 (Manage Relationships) has the following management practices:

- APO08.01 Understand business expectations.
- APO08.02 Identify opportunities, risk and constraints for IT to enhance the business.
- APO08.03 Manage the business relationship.
- APO08.04 Co-ordinate and communicate.
- APO08.05 Provide input to the continual improvement of services.

Inputs and outputs are documented at the management practice level.

COBIT can be freely accessed through www.isaca.org.

TOGAF

The Open Group Architecture Framework, is a framework and method for IT and enterprise architecture practices. TOGAF advocates an "Architecture Development Method" consisting of:

- Architecture Vision
- Business Architecture
- Information Systems Architectures
- Technology Architecture
- Opportunities and Solutions

- Migration Planning
- Implementation, Governance
- Architecture Change Management

TOGAF can be freely accessed through www.opengroup.org.

We will discuss architecture and TOGAF more in Chapter 12.

Other frameworks

Many other frameworks exist, under varying governance models from open to proprietary. An up to date list is maintained by Van Haren Publishing in their publication Global Standards and Publications (Van Haren Publishing, 2016). [Agile frameworks](#) were discussed in Chapter 8. Finally, there is a broad ecosystem of vendor-specific certifications as well, to educate practitioners in the specifics of various commercial products.

Observations on the frameworks

Collaborative

Status: This section is in first draft as of 9 Jun 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

In terms of the new digital delivery approaches, there are a number of issues and concerns with the [frameworks](#):

- The fallacy of statistical process control
- Local optimization temptation
- Lack of execution model
- Proliferation of secondary artifacts, compounded by batch orientation
- Confusion of process definition

Add failure of risk management quote from Hubbard p 73

The problem of statistical process control

CMM author Watts Humphrey's original vision was to apply full statistical process control to the software process. As he stated at the time:

Dr. W. E. Deming, in his work with the Japanese after World War II, applied the concepts of statistical process control to many of their industries. While there are important differences, these concepts are just as applicable to software as they are to producing consumer goods like cameras, television sets, or auto mobiles. ([\[Humphrey1989\]](#), p. 3)

The overall CMM/CMMI idea (in the well-known staged model) is that a process cannot be improved and optimized until it is fully under control. Perhaps well-defined industrial processes should not be optimized until they are fully "managed." However, as we discussed in the previous section, process control theorists see creative, knowledge-intensive processes as requiring [empirical control](#). SPC applied to software has therefore been criticized as inappropriate [\[Racynski2008\]](#).

In CMM terms, empirical process control starts by measuring and immediately optimizing (adjusting). To restate the Martin Fowler quote from the last section: "a process can still be controlled even if it can't be defined."[\[Schwaber2002\]](#) They need not—and **cannot**—be fully defined. One of the most questionable aspects of CMMI therefore, is its implication that process optimization is *something only done at the highest levels of maturity*.

In short, the CMMI staged model encourages the thought that process improvement (optimization) only is possible at Level 5. Many companies implementing CMMI stages however will pragmatically say "Maybe we only need to get to level 3." This implies that they define and manage their processes, but never improve them.

This runs against much current thinking and practice, especially that deriving from Lean philosophy, in which processes are seen as always under improvement. (See discussion of [Toyota Kata](#).) All definition, measurement, and control must serve that end.

The CMMI has evolved since Humphrey's initial vision, but between its mis-applicaton of statistical process control, and the idea that that process optimization is only relevant at the highest maturity, it is (in the view of this author) badly out of step with current digital trends.

The other frameworks do not embrace statistical process control to the same extent as the CMMI. PMBOK suggests that "control charts may also be used to monitor cost and schedule variances, volume, and frequency of scope changes, or other management results to help determine if the project management processes are in control" ([\[PMI2013\]](#), Kindle Locations 4108-4109). This also contradicts the insights of empirical process control, unless the project were also a fully defined process—unlikely from a process control perspective.

Local optimization temptation

We must not seek to optimize every resource in the system ... A system of local optimums is not an optimum system at all; it is a very inefficient system.

— Eli Goldratt, *The Goal*

IT capability frameworks can be harmful if they lead to fragmentation of improvement effort and lack of focus on the flow of IT value.

The digital delivery system at scale is a complex socio-technical system, including people, process, and technology. Frameworks help in understanding it, by breaking it down into component parts in various ways. This is all well and good, but the danger of **reductionism** emerges.

NOTE

There are various definitions of "reductionism." This discussion reflects one of the more basic versions.

A reductionist view implies that a system is nothing but the sum of its parts. Therefore, if each of the parts is attended to, the system will also function well.

This can lead to a compulsive desire to do "all" of a framework. If ITIL calls for 25 processes, then a large, mature organization by definition should be good at all of them. But the 25 processes (and dozens more sub-processes and activities) called for by ITIL, or the 32 called for by COBIT, are somewhat arbitrary divisions. They overlap with each other.

Furthermore, there are many digital organizations that do not use the full ITIL or COBIT process portfolio and yet deliver value as well as organizations that do use ITIL.

This temptation for local, process-level optimization runs counter to core principles of Lean and Systems Thinking. Many management thinkers, including W.E. Deming, Eli Goldratt, and others have emphasized the dangers of local optimization, and the need for taking a systems view.

As this book's structure suggests, delivering IT value requires different approaches at different scales. There is recognition of this among framework practitioners; however, the frameworks themselves provide insufficient guidance on how they scale up and down.

Sutton observations on CMM: stipulates/strongly advocates functional specialization

Lack of execution model

It is also questionable whether even the largest actual IT organizations on the planet could fully implement the frameworks. Specifying too many interacting processes has its own complications.

Consider: Both ITIL and COBIT devote considerable time to documenting possible process inputs and outputs. As a part of every process definition, ITIL has a section entitled "Triggers, inputs, outputs, and interfaces." The Service Level Management Process ([\[TSO2011b\]](#), pp 120-122) for example, lists:

- 7 triggers (e.g. "service breaches")
- 10 inputs (e.g. "customer feedback")
- 10 outputs (e.g. "reports on OLAs")
- 7 interfaces (e.g. "Supplier management")

COBIT similarly details process inputs and outputs. In the Enabling Processes guidance, each management practice suggests inputs and outputs. For example, the APO08 process "Manage Relationships" has an activity of "Provide input to the continual improvement of services," with

- 6 inputs
- 2 outputs

But processes do not run themselves. These process inputs and outputs require staff attention. They often imply [queues](#) and therefore Work in Process, often invisible. They impose demand on the system and each handoff represents transactional friction. Some handoffs may be implemented within the context of an IT management suite; others may require procedural standards, which themselves need to be created and maintained. The industry currently lacks understanding of how feasible such fully elaborated frameworks are, in terms of the time, effort, and organizational structure they imply.

We have discussed the issue of overburden previously. Too many organizations have contending execution models, where projects, processes, and miscellaneous work all compete for people's attention. In such environments, the overburden and wasteful [multi-tasking](#) can reach crisis levels. With ITIL in particular, because it does not cover project management or architecture, we have a very large quantity of potential process interactions that is nevertheless incomplete.

Secondary artifacts, compounded by batch orientation

We move away from heavily documented handoffs to a process that creates only the design artifacts we need to move the team's learning forward.

— Jeff Gothelf, Lean UX

The process handoffs also imply that artifacts (documents of various sorts, models, software, etc.) are being created and transferred in between teams, or at least between roles on the same team with some degree of formality.

Primary artifacts are executable software and any additional content intended directly for value delivery. Secondary artifacts are anything else.

An examination of the ITIL and COBIT process interactions shows that many of the artifacts are secondary concepts such as "plans," "designs," or "reports:"

- Design specifications (high level and detailed)
- Operation and use plan
- Performance reports
- Action plans
- Consideration and approval

and so on. (Note that actually executable artifacts are not included here.)

Again, artifacts do not create themselves. Hundreds of artifacts are implied in the process frameworks. Every artifact implies:

- Some template or known technique for performing it
- People trained in its creation and interpretation
- Some capability to store, version, and transmit it

Unstructured artifacts such as plans, designs, and reports in particular impose high cognitive load. As digital organizations automate their pipelines, it becomes essential to identify the key events and elements they may represent, so that they can be embedded into the automation layer.

Finally, even if a given process framework does not specifically call for waterfall, one can sometimes still see its legacy. For example:

- Calls for thorough project planning and estimation
- Cautions against "cutting corners"
- "Design specifications" moving through approval pipelines (and following a progression from general to detailed)

All of these tend to signal a large batch orientation, even in frameworks making some claim of supporting Agile.

Good system design is a complex process, and we will discuss technical debt in Chapter 12. But the slow [feedback](#) signals resulting from the artifact-centric batch processes implied by some frameworks are unacceptable in current industry. This is in part why new approaches are being adopted.

Confusion of process definition

One final issue with the "process" frameworks is that, while they use the word "process" prominently, they are not aligned with Business Process Management best practices. [\[Betz2011b\]](#)

All of these frameworks provide useful descriptions of major ongoing capabilities and functions that the large IT organization must perform. But in terms of our preceding discussion on process method, they in general are developed from the perspective of steady-state functions, as opposed to a value stream or defined process perspective.

This can be seen by looking above at ITIL, for example. A Business Process Management consultant would see the term "Capacity Management" and observe that it is not countable or event-driven. "How many Capacities did you do today?," might be the question.

The BPM community is clear that processes are countable and event-driven (see [\[Sharp2009\]](#)). Naming them with a strong active verb is seen as essential. "True" IT processes therefore might include:

- Accept Demand
- Deliver Release
- Complete Change

- Resolve Incident
- Improve Service

The lack of countability throughout the IT frameworks' conception of "process" has caused confusion and lack of alignment with BPM professionals for years, and remains an ongoing problem.

In the next chapter section, we'll consider some of these concerns further, and responses consistent with current Agile, Lean, and digital approaches.

Lean, Agile, and IT process management

Collaborative

Status: This section is in first draft as of 9 Jun 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Individuals and interactions over processes and tools.

— Manifesto for Agile Software Development

Like project management, the practice of process management is under scrutiny in the new Lean and Agile-influenced digital world.

There's little question the idea of statistical process control for product development has been superseded by Scrum and Lean Product Development. However, this is not only a textbook on product development. It covers all of traditional IT management, in its new guise of the digitally transformed organization. Agile, per se, is primarily about product (software) development, but one thing the frameworks make clear is that development is only part of digital management.

Clearly, the concept of "process" is important and will persist through digital transformation. It seems we need a more robust set of tools to discuss it. Process control theory may offer some. At a more pragmatic level, this chapter will break the problem into a lifecycle of:

- Process conception
- Process content

- Process execution
- Process improvement

Although we don't preface these topics with "Agile" or "Lean," bringing those and related perspectives to these ideas is the intent of this chapter.

Process conception

"Many companies have at least one dysfunctional area. This may be the "furniture police" who won't let programmers rearrange furniture to facilitate pair programming. Or it may be a purchasing group that takes six weeks to process a standard software order. In any event these types of insanity get in the way of successful projects."

— Mike Cohn, Succeeding with Agile: Software Development Using Scrum

Processes can generate various emotional reactions:

"Dysfunctional! Insanity!" (as above)

"Follow the process!"

"What bureaucracy!"

"Don't create a *Process Police*!"

"I am an ITIL professional. I believe in ITIL!"

"I don't write code on an assembly line!"

Such reactions are commonplace in social media, but we need a more objective and rational approach to understand the pros and cons of processes. Don Reinertsen uses a number of neutral concepts towards this end ([\[Reinertsen2009\]](#), chapters 3, 6 and 7):

- Queues
- Work in Process
- Cadence
- Synchronization
- Sequencing

Many of the issues with process comes from a failure to correctly understand and analyze the organization needs in objective terms such as these. This is the focus of [Toyota Kata](#). A process is a technique, a tool, and no technique should be implemented without a thorough understanding of the business context. Nor should any technique be implemented without rigorous, disciplined follow-up as to its real effects, both direct and indirect.

A skeptical and self-critical, "go and see" approach is therefore essential. Too often, processes are instituted in reaction to the last data point, imposed top down, and rarely evaluated for

effectiveness. Allowing affected parties to lead a process re-design is a core Lean principle (kaizen).

On the other hand, uncoordinated local control of processes can also have destructive effects, as the next section discusses:

The pitfall of process "silos"

One organization enthusiastically embraced process improvement, with good reason: customers, suppliers, and employees found the company's processes slow, inconsistent, and error prone. Unfortunately, they were so enthusiastic that each team defined the work of their small group or department as a complete process. Of course, each of these was in fact the contribution of a specialized functional group to some larger, but unidentified, processes. Each of these "processes" was "improved" independently, and you can guess what happened.

Within the boundaries of each process, improvements were implemented that made work more efficient from the perspective of the performer. However, these mini-processes were efficient largely because they had front-end constraints that made work easier for the performer but imposed a burden on the customer or the preceding process. The attendant delay and effort meant that the true business processes behaved even more poorly than they had before. This is a common outcome when processes are defined too "small." Moral: Don't confuse subprocesses or activities with business processes.

— Alex Sharp, Workflow Modeling

obtain permission

The above quote well illustrates the dangers of combining local optimization and process management. Many current authors speak highly of self-organizing teams, but self-organizing teams may seek to optimize locally. Process management was originally intended to overcome this problem, but modeling techniques can be applied at various levels, including within specific departments. This is where enterprise business architecture can assist, by identifying these longer, end to end flows of value and highlighting the handoff areas, so that the process benefits the larger objective.

Process proliferation

Another pitfall we cover here is that of process proliferation.

Process is a powerful tool. Ultimately it is how value is delivered. However, too many processes can have negative results on an organization.

Review the fundamentals of process modeling, and the discussion of the frameworks above. One thing that you do not see is any attention to the resource impacts of the process. This is a primary difference between project and process management; in process management (both theory and frameworks), resource availability is in general assumed.

More advanced forms of process modeling and simulation (see "[discrete event simulation](#)") can provide insight into the resource demands for processes. However, such techniques 1) require specialized tooling and 2) are not part of the typical BPM practitioner's skillset.

Many enterprise environments have multiple cross-functional processes:

- service requests,
- compliance certifications,
- asset validations,
- provisioning requests,
- capacity assessments,
- change approvals,
- training obligations,
- performance assessments,
- audit responses,
- expense reporting,
- travel approvals,

and many more.

Such processes can be implemented on the assumption that enterprises can always accommodate another process. The result can be dramatic overburden for digital staff in complex environments.

It is therefore advisable to at least keep an inventory of processes that may impose demand on staff, and understand both the aggregate demand as well as the degree of [multi-tasking and context-switching](#) that may result (as discussed in Chapter 5). Thorough automation of all processes to the maximum extent possible can also drive value, as both load and distraction are reduced.

Process content

Sometimes, the process simply becomes too complex for humans to follow. Multi-page process models are rarely consulted, even if the team affected developed them. This is especially true when there are many optional or conditional steps in the process.

There are two related lines of response to this:

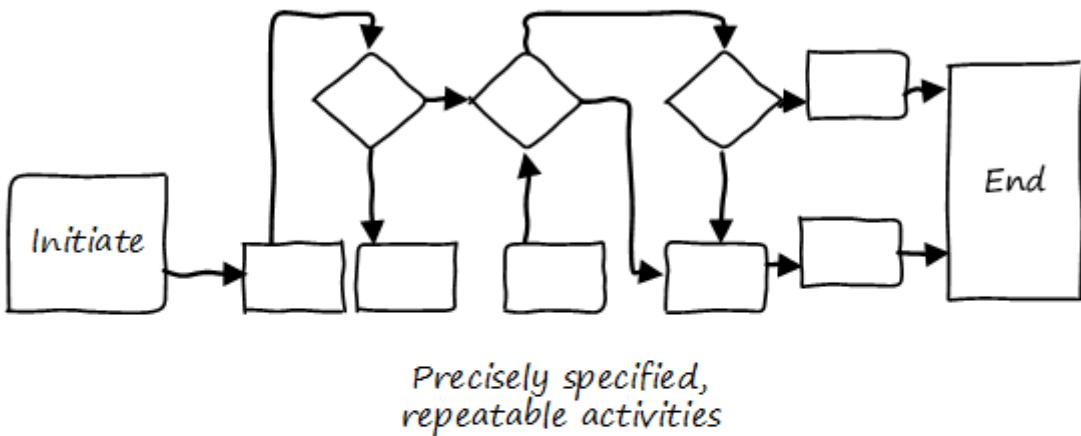
- Case Management
- The Checklist Manifesto

Case Management

NOTE Do not confuse "Case" here with Computer Assisted Software Engineering.

Case management is a concept used in medicine, law, and social services. Case management can be thought of as a high-level process supporting the skilled knowledge worker applying their professional expertise.

Process management



Case management



To quote the Workflow Management Coalition:

[Business Process Modeling] and [Case Management] are useful for different kinds of business situations.

- Highly predictable and highly repeatable business situations are best supported with BPM.
 - For example signing up for cell phone service: it happens thousands of times a day, and the process is essentially fixed.
- Unpredictable and unrepeatable business situations are best handled with ACM.
 - For example investigation of a crime will require following up on various clues, down various paths, which are not predictable before hand. There are various tests and procedures to use, but they will be called only when needed. [\[WFMC2010\]](#), via [\[England2013\]](#)

Noted IT consultant and author Rob England contrasts "case management" with "standard process" in his book *Plus! The Standard+Case Approach: See Service Response in a New Light*. Some processes are repeatable and can be precisely standardized, but it is critical for anyone working in complex environments to understand the limits of standard process.

The Checklist Manifesto

The Checklist Manifesto is the name of a notable book by author/surgeon Atul Gawande [\[Gawande2010\]](#). The title can be misleading; the book in no way suggests that all work can be reduced to repeatable checklists. Instead, it is another in depth examination of the relationship between standardization and complexity. Like case management, it addresses the problem of complex activities requiring professional judgement.

Unlike case management, it explores more time-limited and often urgent activities such as flight operations, large scale construction, and surgery. These activities, as a whole, cannot be reduced to one master process; there is too much variation and complexity.

However, within the overall bounds of flight operations, or construction, or surgery, there are critical sequences of events that MUST be executed, often in a specific order. Gawande discusses the airline industry as a key exemplar of this. Instead of one "master checklist" there are specific, clear, brief checklists for a wide variety of scenarios, such as a cargo hold door becoming unlatched.

There are similarities and differences between core BPM approaches and checklists. Often, business process management is employed to describe processes that are automated and whose progress is tracked in a database. Checklists, on the other hand, may be more manual, intended for use in a closely collaborative environment (such as a aircraft cockpit or operating room), and may represent a briefer period of time.

Finally, in chapter 8, we discussed the concept of the [submittal schedule](#), which Gawande encountered in his discussions with the construction industry. In this approach, the actual, trackable task is that **two or more parties consult with each other**. This technique is also applicable in a BPM and Case Management sense.

Process execution

Since our initial discussions in Chapter 5 on Work Management, we have progressed through Project and Process Management, and find ourselves returning full circle. Despite the various ways in which work is conceived, funded, and formulated, at the end "it's all just work." We must retain our concern for the "human resources" (that is, people) who find themselves at the mercy of:

- project fractional allocations driving multi-tasking
- processes imposed top down with no demand analysis or evaluation of benefits
- myriad demands that, although critical, do not seem to fit into either of the first two categories

The Lean movement manages through minimizing waste and over-processing. This means both taking out un-necessary steps from processes, **AND eliminating un-necessary processes completely when required.** Correspondingly, the processes that remain should have high levels of visibility. They should be taken with the utmost seriousness, and their status should be central to most people's awareness. (This is the purpose of [Andon](#).)

From workflow tools to collaboration and digital exhaust

One reason process tends to generate friction and be unpopular is the poor usability of workflow tools. Older tools tend to present myriads of data fields to the user and expect a high degree of training. Each state change in the process is supposed to be logged and tracked by having someone sign in to the tool and update status manually.

By contrast, modern workflow approaches take full advantage of mobile platforms and integration with technology like chat rooms. Mobile development imposes higher standards for user experience (UX) design, which makes tracking workflow somewhat easier. Integrated software pipelines that integrate application lifecycle management and/or project management with source code and build management are increasingly gaining favor. For example:

1. A user logs a new feature request in the ALM tool
2. When the request is assigned to a developer, the tool automatically creates a feature branch in the source control system for the developer to work on
3. The developer writes tests and associated code that merge changes back to the central repository once tests are passed successfully
4. The system automatically runs build tests
5. The ALM tool is automatically updated accordingly with completion if all tests pass

In an operations context, certain operational commands can be executed directly from the chat window. This is known as "ChatOps:"

While in a chat room, team members type commands that the chat bot is configured to execute through custom scripts and plugins. These can range from code deployments to security event responses to team member notifications. The entire team collaborates in real-time as commands are executed. [\[Sigler2014\]](#)

Properly configured ChatOps provides a rich audit trail of who did what, when, and who else was involved. Fundamental governance objectives of accountability can be considered fulfilled in this way, on par with paper or digital forms routed for approval (and with corresponding delays).

In general, the idea is that we can understand digital processes not through painful manual status updates, but rather through their digital exhaust - the data byproducts of people performing the value-add day to day work, at speed and with flow instead of constant delays for approvals and status updates.

Process improvement

There tended to be no big picture waiting to be revealed . . . there was only process kaizen . . . focused on isolated individual steps. . . . We coined the term “kamikaze kaizen” . . . to describe the likely result: lots of commotion, many isolated victories . . . [and] loss of the war when no sustainable benefits reached the customer or the bottom line.

— Womack and Jones, Lean Thinking

Finally, what is there to say about process improvement from an Agile and Lean perspective? Since these movements are founded on improvement cycles, fast feedback, and self-organization, it might seem redundant to have a section on Agile Process Improvement.

But dysfunction and antipatterns exist here:

- Not basing process improvement in an empirical understanding of the situation
- Process improvement activities that do not involve those affected
- Not treating process activities as demand in and of themselves
- Uncoordinated improvement activities, far from the bottom line

The solutions are to be found largely within Lean theory.

- Understand the facts of the process; do not pretend to understand based on remote reports. "Go and see," in other words.
- Respect people, and understand that best understanding of the situation is held by those closest to it.
- Make time and resources available for improvement activities. For example, assign them a Problem ticket and ensure there are resources specifically tasked with working it, who are given relief from other duties.
- Periodically review improvement activities as part of the overall portfolio. You are placing "bets" on them just as with new features. Do they merit your investment?

Topics in process management

Collaborative

20160609 Status: Like most topics sections, this one will remain rough until 2nd draft.
Suggestions appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

The idea of PMDB and PIs

Originally published at <http://www.lean4it.com/2014/06/in-search-of-the-pcmdb-and-pcis.html>.

In search of the PMDB and PIs

As I was preparing chicken piccata last night, I had one of those flashes.

In the IT industry, why don't we care about process information the same way we care about configuration information?

We've had a 15-year conversation about CMDBs. It's a well known idea that all configuration data should be put in a repository, or short of that, should be seen as part of a federated system. Applications, servers, databases, files, queues, executables, services, racks, switches, etc, etc, etc should all be seen as Configuration Items, or "CIs."

But, if you ask ITSM advocates about processes, they will talk specifically about Incidents, Problems, Changes, Releases, etc as if they all are fundamentally different.

I think we need the concept of a Process Item - just as a specific Server and an Application can both be CIs, so can a specific Incident and a Change both be PIs. They could inhabit the same PMDB, or be federated in a common PMS. [Yes, I recognize the unfortunate acronym :-) Let's figure out better naming later.]

This in fact is how many actual implementations of IT management tools work. But at a framework level, it is NOT how people are educated in IT management - the education and mental model continues to be that Incidents, Changes and Problems are very distinct and at best cross-reference each other.

In the world of Process, we have a series of related terms that indicate a "potentially recurring series of events with various complexity (parallelism, sequentially

related or not, and so on)"

as the Oracle Communications Model states. Under this general heading we have:

- Program
- Project
- Release
- Change
- Service Request
- Incident
- Problem (and other Improvement Opportunity)

But (as I have pointed out elsewhere) the way these are currently implemented is too often as distinct queues of work with no coordination and understanding of aggregate demand. There is no awareness or concern when people who are critical to a Project also wind up handling high criticality Incidents.

I think there is much justification for promoting the ideas of Process Item and a unified PMDB as industry guidance.

First, it aligns well with the growing interest in Kanban and limiting work in process. All these Process types compete for resources. If all are seen as fundamentally similar then it is possible to understand the aggregate demand.

My coverage as an analyst of the Professional Services Automation tool sector (e.g. Tigerpaw) suggests that this class of tooling is already moving in this direction.

There are complex dependencies between PIs just as with CIs - larger grained PIs such as Project may contain any number of smaller grained PI types such as Release, Change, and Request.

The above-mentioned Oracle Communications Model (which is now conformant with eTOM) requires that the PROCESS entity be used as the overarching parent for all process activity.

During one large program I was part of, all Releases, Project milestones, and Changes were consolidated into a single timeline view which was praised by senior executives as one of the best orienting mechanisms they had seen. I have seen various attempts and interest along these lines, but always company or vendor specific. I wonder if the lack of a PI/PMDB concept in fact may be why the concept of CMDB has had such struggles.

People have often said to me, "Isn't the CMDB the basis of ERP for IT"?

My view is no, no more than the Bill of Materials is the basis for manufacturing ERP. The core of ERP systems is more about process and execution, and that is what we are missing in large scale IT management. We've spent years struggling with the data-centric orientation the CMDB discussion gave us, while shortchanging the process and execution side of the work.

I realize the red flags that come up with any suggestion of a centralized system, which is why

I am more in favor of seeing the PI concept as a logical interface. If all Process Items had some degree of overarching consistency, they could be exchanged and federated as part of execution activities, without requiring a monolithic centralized data store.

Thoughts?

===== Management cadencing - e.g. kanban

===== ITSM core processes

- Incident, Change, etc

===== KPIs and metrics

===== The shared services problem

Lifecycle: collaboration, internal commoditization, external commoditization
Effectiveness, efficiency, governance

Examples of shared services for digital teams - layered nature

Service catalog: act, offering, system

Infrastructure vs business

Benchmarkable

Deeper discussion of virtual server provisioning as example (backref to ch2, forward to ch10)

example of service-sized process dysfunction

Adapting a services model to improve flow

(Notice that the services a functional area provides can be viewed as a form of product in and of themselves.)

Degree of automation perils of "take a number" service-ization

- SLAs and the service catalog
 - Service-izing functions: pros and cons

Scenario: repetitive yet error prone work team to team interactions that become "servicized"

Implication in various IT governance texts that the shared services team is accountable for optimal use of the shared resource. A very dangerous guideline when multiplied by numerous COEs/SSs.

The folly of local process optimization... CRITICAL to look at process in context

===== The BPM community

===== Tooling

BPMN, BPEL, and more

simulations: systems dynamics & DES

ITSM tooling: process PLUS what is being acted on (portfolio)

===== The ERP problem
Where IT management diverges from Agile

===== ISO 9000 <http://www.iso.org/iso/pub100080.pdf>

===== Digital services playbook <https://playbook.cio.gov/#play10>

===== Deeper theory
Cynefin
"best practice"
Cost of delay of aggregate process - research idea

formalisms state charts, Petri nets

Process as a Mintzberg liaison mechanism

http://blogs.forrester.com/jean_pierre_garbani/09-06-26-my_issue_itil

CPRET is a mnemonic for the basic definition of process in process engineering: it stands for Constraints, Product, Resources, input Elements and Transformation which are the basic components of a process. In process engineering, a process is a suite of transformations of elements into a given output (product) given a set of constraints and resources. From this definition, we can see that technology has a strong influence on the process: the transformation part is a clear function of the technology available as input and resources in IT are strongly influenced by the technology used. As we mostly deal with information and data in IT management processes, the type of data available is either helping or impeding the transformation part.

Schwaber process author - chem e guy

Kennelay - PID model

formal process control

the emergence of governance and assurance - Standards and processes are meaningless without assurance. (chapt 10 foreshadow)

Sidebar: Lessons from manufacturing, operations research, etc

Controversy of application to IT

Will this have been foreshadowed?

review & incorporate more grounding from

<http://www.computing.dcu.ie/~mcrane/CA441/BPMLectureNotes.pdf> [BPM Lecture Notes]

Process capability is the inherent variation of the process in relation to the specification limits. The smaller the process variation, the better the process's capability. <<Kan1995>>

http://jeffreymbradshaw.net/publications/Common_Ground_Single.pdf & allspaw commentary - the basic questions underlying various current arguments such as "DevOps v ITIL" and "No Projects"

Conclusion

In conclusion, we have considered:

- Why process management matters, and is not likely to go away
- The basics of process description and modeling
- Some background and history of continuous improvement and process control
- The major IT-related process frameworks
- Emerging approaches for digital process management.

As you stand on the threshold of scaling your company yet again, to the enterprise level, you will find process management an essential tool. Like any powerful tool, it has its dangers if misused. Be wary of claims of statistical process control in creative activities, and avoid burdensome process tracking and compliance approaches. Challenge all proposed processes, keep them lean, but if they "make the cut," then make sure everyone knows their existence and status.

Discussion questions

1. What processes do you experience daily? Weekly?
2. Review the section on the [Spotify model](#). How does it relate to the discussion here?
3. Sometimes, organizations try to treat a complex process as defined, instead of managing it empirically. Sometimes, people react by calling this "Taylorism." Why? Google and discuss.
4. Have you experienced a problem where an improved process would have helped? What about a problem where process seemed to be the cause of it?

Research & practice

1. Compare and contrast the Scaled Agile Framework and COBIT.
2. Research BPMN notation and use it to document a process familiar to you
3. Research process simulation and prepare a report. Optionally, compare process simulation with systems dynamics modeling.
4. We know that completely defined processes can be placed under statistical control, and that creative processes cannot be. What about processes falling between these two extremes, such as help desk call management? What does the research say about statistical control of such processes?

Further reading

The definitive BPM reference is [\[Rummel1995\]](#). Good practical applications are found in [\[Burlton2001\]](#), [\[Harmon2003\]](#), and [\[Sharp2009\]](#).

The history of Scrum is relevant, in particular [\[Schwaber2002\]](#).

Lean theory is also essential, e.g. [\[Womack2003\]](#), [\[Rother2010\]](#).

Critiques of statistical process control for software are found in [\[Binder1997\]](#) and [\[Racynski2008\]](#).

Books

- [Improving Performance: How to Manage the White Space in the Organization Chart](#)
- [Business Process Change: A Guide for Business Managers and BPM and Six Sigma Professionals](#)

Articles

- [Business Process Management: A Comprehensive Survey](#)
- [Why “Enterprise DevOps” Doesn’t Make Sense](#)
- [BPM Lecture Notes](#)

Videos

- [Geary Rummler in 1981](#)

Professional

- [BPTrends: BPM Analysis, Opinion, and Insight](#)
- [Process Renewal Group Articles and Resources](#)

Section conclusion

It's been an interesting ride, scaling up from a team to a team of teams.

We've considered our organizational structure and our culture, and the importance of culture in organizational effectiveness.

We've also considered the fundamentals of project and resource management, with some caution and with an understanding that not all work is performed through classical project management and its iron triangle of cost, quality, and scope.

Finally, process management is a critical practice in business operations, and will continue into the digital era. There are a variety of approaches, concerns and techniques that must be considered and correctly applied.

Section III is essential prep to Section IV - essentially, section III is about management, where section IV is about governance

Section IV: Enterprise

Collaborative

Status: This section is in active development as of Jun 24 2016. Expect volatility.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Section IV introduction

Scenario

You are now running one of the larger and more complex IT-based operations on the planet, with an annual IT budget of hundreds of millions or billions of dollars. You have thousands of programmers, systems engineers, and IT managers, with a wide variety of responsibilities. IT is in your market-facing products and in your back-office operations. In fact, it's sometimes hard to distinguish the boundaries as your company transforms into a digital business.

Agile techniques remain important to you, but things are getting complex and you're testing the boundaries of what is possible. How can you operate at the scale you've achieved and still be Agile? As usual in life, you're finding that there are always tradeoffs. Decisions you made long ago come back to haunt you, security threats are increasing, and at your scale there's no escaping the auditors.

You have scaled up in terms of size, what is less often understood that scaling up in size also means scaling up in terms of timeframes: concern for the past and the future extend further and further in each direction. Organizational history is an increasing factor, and the need to manage this knowledge base can't be ignored.

But you have great resources at your command, and you're as well positioned as any of your competitors to meet the challenges ahead. And in the end, that's all you need.

Chapter 10: Governance, Risk, Security, and Compliance

We need to cope with new layers of enterprise organization, and external forces (regulators, vendor partners, security adversaries, auditors) increasingly defining our options. This chapter sets

the frame for the section. Chapters 11 and 12 in many ways are further elaborations of two major domains of governance concerns. [rewrite]

Chapter 11: Enterprise Information Management

We've been concerned with data, information, and knowledge since the earliest days of our journey. But at this scale, we have to formalize our approaches and understandings; without that, we will never capture the full value available with modern analytics and Big Data.

Looking inward, we need to measure this massive IT estate and understand it as an overall dynamic and complex system.

Chapter 12: Architecture and Portfolio

We need to understand the big picture of interacting lifecycles, reduce technical debt and redundancy, and obtain better economies of scale. We need to define our investment strategy based on a sound understanding of both business needs and technology limitations.

Special section: The IT lifecycles

Collaborative

Status: This section is in active development as of Jun 24 2016. Expect volatility.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

We've discussed products and the various ways digital organizations deliver them, from simple work management to more sophisticated project and process management approaches. Now, we need to refine our understanding of the products themselves and how they are managed.

We previously discussed the relationship between [feature versus component teams](#) in chapter 4. In chapter 9, we touched on the idea of [shared services](#) teams. Both of these ideas are now expanded into what is called the "four lifecycle model."

The four lifecycle model was first documented in [\[Betz2011a\]](#). The four lifecycles are:

- The Application Service lifecycle
- The Infrastructure Service lifecycle
- The Asset lifecycle
- The Technology Product lifecycle

Each of these lifecycles reflects the existence of a significant concept, that is managed over time, as a portfolio. (More on IT portfolio management practices in Chapter 10.)

First, bear in mind that services are equivalent to products. Digital value is usually delivered as a service, and therefore shares standard service characteristics from an academic perspective, including the idea that services are produced and consumed simultaneously (e.g. an account lookup) and are perishable (a computer's idle time cannot be recovered if it goes unused).

The first two concepts (Application and Infrastructure Service) below reflect these characteristics; the second two (Asset and Technology Product) do not.

An **Application Service** is a business or market-facing product, consumed by people whose primary activities are **not** defined by an interest in [information technology](#): for example, a bank customer looking up her account balance, or an Accounts Payable systems operator. In terms of "feature versus component," the concept of Application is more aligned to "feature." An example would include an Online Banking system or a Payroll system.

The **Application Service Lifecycle** is the end to end existence of such a system, from [idea to retirement](#). In general, the realization such a system is needed originates **externally** to the IT capability (regardless of its degree of centralization.) Software as a Service usage is also tracked here.

An **Infrastructure Service** is, by contrast and as [previously discussed](#), a digital or IT service primarily of interest to other digital or IT services/products. Its lifecycle is similar to that of the application service, except that the user is some other IT service. An example would be a Storage Area Network system managed as a service, or the integrated networking system required for connectivity in a data center. Platform and Infrastructure as a Service is also tracked here.

Note that in terms of our , the above lifecycle concepts are service **systems**. The lifecycle of service offerings is a business lifecycle having more to do with go to market strategy on the part of the firm. We covered this to some extent in Chapter 4 and will revisit it in Chapter 12.

An **Asset** is a valuable, tangible investment of organizational resources that is tracked against loss or misuse, and optimized for value over time. It can sit unused and still have some value. Examples would include a physical server or other device, or a commercial software license. Whether assets can be virtual is a subject of debate and specific to the organization's management objectives (Given the licensing implications of virtual servers, treating them as assets is not uncommon.)

The **Asset Lifecycle** is distinct from the service lifecycles, following a rough order including standard supply chain activities:

- Forecast
- Requisition

- Request quote
- Order
- Deliver
- Accept
- Install/configure
- Operate
- Dispose

A contract reserving Cloud capacity is also an Asset.

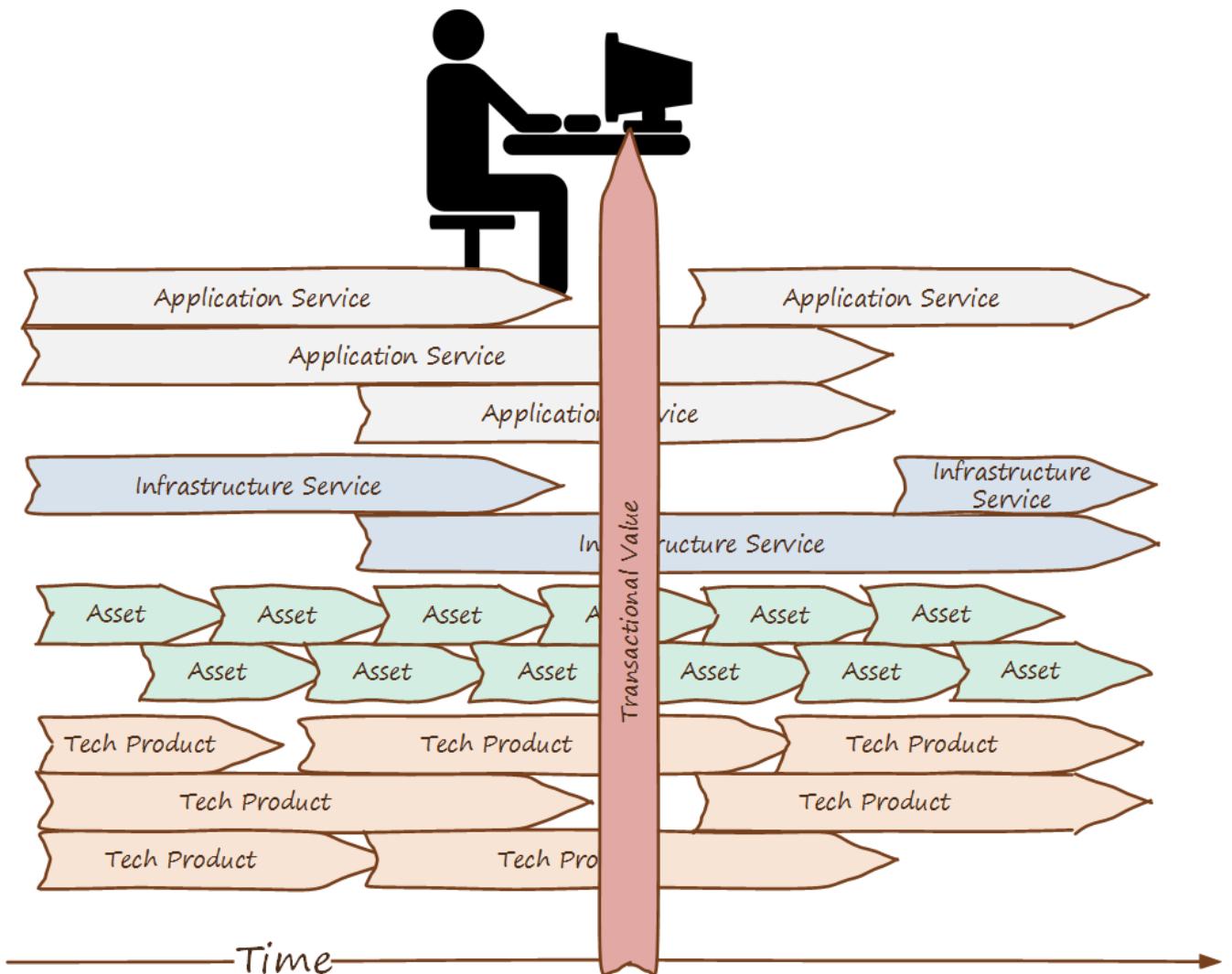
Finally, a **Technology Product** is a **class** of Assets, the "type" to the Asset "instance." For example, the enterprise might select the Oracle relational database as a standard Technology Product. It might then purchase 10 licenses, which are Assets.

The **Technology Product Lifecycle** is also distinct from both the Service and Asset lifecycles:

- Identify technical requirement or need
- Evaluate options
- Select product (may kick off Asset Lifecycle)
- Specify acceptable use
- Maintain vendor relationship
- Maintain product (e.g. patching and version upgrades)
- Continuously evaluate product's fitness for purpose
- Retire product from environment

Cloud services need to be managed in terms of what version they are and what the interoperability concerns are.

The challenge in digital management is "lining up the lifecycles" so that transactional value flows across them.



This can be very difficult, as each lifecycle has a logic of its own, and there may be multiple interdependencies. A technology product may come to the end of its market life and drive expensive changes up the stack. Conversely, new application requirements may expose deficiencies in the underlying stack, again requiring expensive remediation. Technology product vulnerabilities can prove disruptive, and the asset lifecycle (representing either physical depreciation and refresh cycles, or time-bound licensing) is a significant cost driver.

These lifecycles are essential to all of the next 3 chapters.

- In Chapter 10, they represent focal points for risk and control
- In Chapter 11, we will see how enterprise information management depends on their management
- In Chapter 12, we will further discuss how they are managed in terms of architecture and portfolio.

Chapter 10: Governance, Risk, Security, and Compliance

Collaborative

Status: Chapter 10 is in first draft as of 13 July 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Introduction to Chapter 10

Operating at scale requires a different mindset. When you were starting out, the horizon seemed bounded only by your imagination, will, and talent.

At enterprise scale, it's a different world. You find yourself constrained by indifferent forces and hostile adversaries, some of them competing fairly, and others seeking to attack by any means.

Whether or not you are a for-profit, publicly traded company, you are now large enough that audits are required; you also likely have directors of some nature. Like it or not, the concept of "controls" has entered your awareness.

As a team of teams, you needed to understand resource management, finance, the basics of multiple product management and coordination, and cross-functional processes. Now that you are an enterprise, you need also to consider questions of corporate governance. Your stakeholders have become more numerous and their demands have multiplied, so the well-established practice of establishing a governing body has been applied.

Security threats increase proportionally to the company's size. The talent and persistence of these adversaries is remarkable. Other challenging players are, on paper, "on the same side," but auditors are never to be taken for granted. Why are they investigating IT systems? What are their motivations and responsibilities? Finally, what laws and regulations are relevant to IT?

IMPORTANT

As with other chapters in the later part of this book, we are going to some degree introduce this topic "on its own terms." We will then add additional context and critique in subsequent sections.

More than any other chapter, the location of this chapter (and especially its Security subsection) in

Section 4 draws attention. Again, any topic in any chapter may be a matter of concern at any stage in an organization's evolution.

You've been doing security since your company started, otherwise you would not have gotten this big. But now, you have a Chief Information Security Officer, formal risk management processes, a standing director-level security steering committee, auditors, and compliance specialists. That kind of formalization does not usually happen until an organization grows to a certain size.

We needed the content in Section 3 to get this far. We had to understand our structure, how we were organizing our strategic investments, and how we were engaging in operational activities. In particular **it's difficult for an organization to govern itself without some ability to define and execute processes, as processes often support governance controls and security protocols.**

This chapter covers "Governance, Risk, Security, and Compliance" because there are clear relationships between these concerns. They have important dimensions of independence as well. It is interesting that Shon Harris' popular Guide to the CISSP starts its discussion of security with a chapter titled "Information Security Governance and Risk Management." Governance leads to a concern for risk, and security specializes in certain important classes of risk. Security requires grounding in governance and risk management.

Compliance is also related but again distinct, as the concern for adherence to laws and regulations, and secondarily internal policy.

Chapter 10 outline

- Governance
- Enablers
- Risk management
- Compliance
- Assurance and audit
- Security
- Digital Governance

Chapter 10 learning objectives

- Define governance versus management
- Describe key objectives of governance according to major frameworks
- Define risk management and its components
- Describe and distinguish assurance and audit, and describe their importance to digital operations
- Discuss digital security concerns and practices
- Identify common regulatory compliance issues
- Describe how governance is retaining its core concerns while evolving in light of digital transformation

- Describe automation techniques relevant to supporting governance objectives throughout the digital delivery pipeline

Governance

Collaborative

Status: This section is first draft as of 27 June 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

What is governance?

The system by which organizations are directed and controlled.

— Cadbury Report

The COBIT 5 framework makes a clear distinction between governance and management. These two disciplines encompass different types of activities, require different organisational structures and serve different purposes . . . In most enterprises, governance is the responsibility of the board of directors under the leadership of the chairperson [while] management is the responsibility of the executive management under the leadership of the CEO.

— COBIT 5 Framework, ISACA

To talk about digital or IT governance, we must talk about governance in general.

Governance is a challenging and often misunderstood concept. First and foremost, it must be distinguished from "management." This is not always easy, but remains essential.

A governance example

Here is simple explanation of governance:

Suppose you own a small retail store. For years, you were the primary operator. You might have hired an occasional cashier, but that person had limited authority; they had the keys to the store and cash register, but not the safe combination, nor was their name on the bank account. They did not talk to your suppliers. They received an hourly wage and you gave them direct and ongoing supervision.

Credit: I got this analogy from hearing Brian Barnier speak at an ISACA meeting around 2011.

In this case, you were a manager. Governance was not part of the relationship.



Figure 4. Someone to "mind the store" [15: image credit

<https://www.flickr.com/photos/garryknight/11240024613>, commercial use permitted.]

Now, you wish to go on an extended vacation - perhaps a cruise around the world, or a trek in the Himalayas. You need someone who can count the cash and deposit it, and place orders with and pay your suppliers. You need to hire a professional manager.

They will likely draw a salary, perhaps some percentage of your proceeds, and you will not supervise them in detail as you did the cashier. Instead, you will set overall guidance and expectations for the results they produce. How do you do this? And perhaps even more importantly, how do you trust this person?

Now, you need governance.

As we see in the above quote, one of the most firmly reinforced concepts in the COBIT guidance (more on this and ISACA in the next section) is the need to distinguish governance from management. Governance is by definition a board-level concern. Management is the CEO's concern.

In this distinction, we can still see the shop owner and his or her delegate.

IMPORTANT

There is too often a temptation to lump all of "management" in with governance. Sometimes it may be said that the VP of sales, or HR, "governs" their function, for example. While tempting to executives who want to elevate their status, this is not the intent of the term, as we will detail below.

Some theory of governance

In political science and economics, the need for governance is seen as an example of the [principal-agent problem \[Eisenhardt1989\]](#). Our shopkeeper example illustrates this. The hired manager is the "agent," acting on behalf of the shop owner, who is the "principal."

In principal-agent theory, the agent may have different interests than the principal. The agent also has much more information (think of the manager running the shop day to day, versus the owner off climbing mountains).

The agent is in a position to do economic harm to the principal; to shirk duty, to steal, to take kickbacks from suppliers. Mitigating such conflicts of interest is a part of governance.

In larger organizations (such as you are now), it's not just a simple matter of one clear owner vesting power in one clear agent. The corporation may be publicly owned, or in the case of a non-profit, it may be seeking to represent a diffuse set of interests (e.g. environmental issues). In such cases, a group of individuals (directors) is formed, often termed a "board," with ultimate authority to speak for the organization.

The principal-agent problem can be seen at smaller scale within the organization. Any manager encounters it to some degree, in specifying activities or outcomes for subordinates. But this does not mean that the manager is doing "governance," as governance is by definition an organization-level concern.

The fundamental purpose of boards of directors and similar bodies is to take the side of the principal. This is easier said than done; boards can become overly close to an organization's senior management - the senior managers are real people, while the "principal" may be an amorphous, distant body of shareholders and/or stakeholders.

Because governance is the principal's concern, and because the directors represent the principal, governance, including IT governance, is a board-level concern.

There are various principles of corporate governance we will not go into here, such as shareholder rights, stakeholder interests, transparency, and so forth. References on these topics are included in the chapter conclusion (COSO and ISACA are good places to start).

However, as we turn to our focus on digital and IT-related governance, there are a few final insights from principal-agent theory that are helpful to understanding governance. Consider:

the heart of principal-agent theory is the trade-off between (a) the cost of measuring behavior and (b) the cost of measuring outcomes and transferring risk to the agent. [Eisenhardt1989]

What does this mean?

Suppose the shopkeeper tells the manager, "I will pay you a salary of \$50,000 while I am gone, assuming you can show me you have faithfully executed your daily duties."

The daily duties are specified in a number of checklists, and the manager is expected to fill these out daily and weekly, and for certain tasks, provide evidence they were performed (e.g. bank deposit slips, checks written to pay bills, photos of cleaning performed, etc.). That is a behavior-driven approach to governance. The manager need not worry if business falls off; they will get their money. The owner has a higher level of uncertainty; the manager might falsify records, or engage in poor customer service so that business is driven away. A fundamental conflict of interest is present; the owner wants their business sustained, while the manager just wants to put in the minimum effort to collect the \$50,000.

When agent responsibilities can be well specified in this manner, it is said they are highly *programmable*.

Now, consider the alternative. Instead of this very scripted set of expectations, the shopkeeper might tell the manager, "I will pay you 50% of the shop's gross earnings, whether they may be. I'll leave you to follow my processes however you see fit. I expect no customer or vendor complaints when I get back."

In this case, the manager's behavior is more aligned with the owner's goals. If they serve customers well, they will likely earn more. There are any number of hard-to-specify behaviors (*less programmable*) that might be highly beneficial.

For example, suppose the store manager learns of an upcoming street festival, a new one that the owner did not know of or plan for. If the agent is managed in terms of their behavior, they may do nothing—it's just extra work. If they are measured in terms of their outcomes, however, they may well make the extra effort to order merchandise desirable to the street fair participants, and perhaps hire a temporary cashier to staff an outdoor booth, as this will boost store revenue and therefore their pay.

(Note that we have considered similar themes in our discussion of [Agile and contract management](#), in terms of risk sharing.)

In general, it may seem that an outcome-based relationship would always be preferable. There is, however, an important downside. It transfers risk to the agent (e.g. the manager). And because the agent is assuming more risk, they will (in a fair market) demand more compensation. The owner may find themselves paying \$60,000 for the manager's services, for the same level of sales, because the manager also had to "price in" the possibility of poor sales and the risk that they would only make \$35,000.

Finally, there is a way to align interests around outcomes without going fully to performance-based pay. If the manager for cultural reasons sees their interests as aligned, this may mitigate the principal-agent problem. In our example, suppose the store is in a small, tight-knit community with a strong sense of civic pride and familial ties.

Even if the manager is being managed in terms of their behavior, their cultural ties to the community or clan may lead them to see their interests as well aligned with those of the principal.

As noted in [Eisenhardt 1989], "Clan control implies goal congruence between people and, therefore, the reduced need to monitor behavior or outcomes. Motivation issues disappear." We have discussed this kind of motivation in Chapter 7, especially in our discussion of **control culture** and insights drawn from the military.

COSO and control

Internal control is a process, effected by an entity's board of directors, management, and other personnel, designed to provide reasonable assurance regarding the achievement of objectives relating to operations, reporting, and compliance.

— Committee of Sponsoring Organizations of the Treadway Commission, Internal Control - Integrated Framework

An important discussion of governance is found in the statements of COSO on the general topic of "control."

Control is a term with broader and narrower meanings in the context of governance. In the area of risk management, "controls" are specific approaches to mitigating risk. However, "control" is also used by COSO in a more general sense to clarify governance.

What is COSO?

The Council of Sponsoring Organizations of the Treadway Commission (COSO) has a non-intuitive name, especially given its global influence.

COSO is a "private sector initiative," funded by:

- Institute of Certified Public Accountants (AICPA),
- American Accounting Association (AAA),
- Financial Executives International (FEI),
- Institute of Internal Auditors (IIA)
- Institute of Management Accountants (IMA).

It was founded in 1985 to support the National Commission on Fraudulent Financial Reporting, and has published various reports and guidance mostly concerned with the topic of internal control.

Control activities, according to COSO, are

the actions established through policies and procedures that help ensure that management's directives to mitigate risks to the achievement of objectives are carried out. Control activities are performed at all levels of the entity, at various stages within business processes, and over the technology environment. They may be preventive or detective in nature and may encompass a range of

manual and automated activities such as authorizations and approvals, verifications, reconciliations, and business performance reviews.

...Ongoing evaluations, built into business processes at different levels of the entity, provide timely information. Separate evaluations, conducted periodically, will vary in scope and frequency depending on assessment of risks, effectiveness of ongoing evaluations, and other management considerations. Findings are evaluated against criteria established by regulators, recognized standard-setting bodies or management and the board of directors, and deficiencies are communicated to management and the board of directors as appropriate. [COSO2013]

Systems theory, feedback, control, and governance We've encountered systems theory and associated concepts such as **feedback** and **control** throughout this book. The idea of "governance" predates these, but in an interesting way.

The term "governance" originates from the Greek word κυβερνάω [kubernáo], which means "to steer," as in a ship. Nautical navigation is a process of feedback and correction. The same Greek word is also the basis for the term "cybernetics," another word closely associated with systems and control theory.

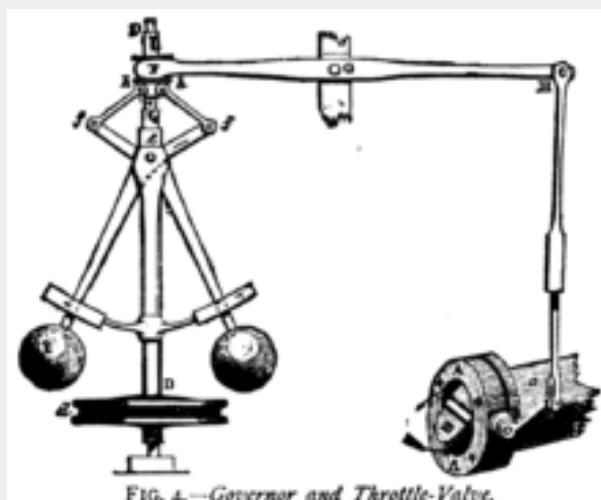


Figure 5. Centrifugal governor [16: Public domain image via Wikipedia]

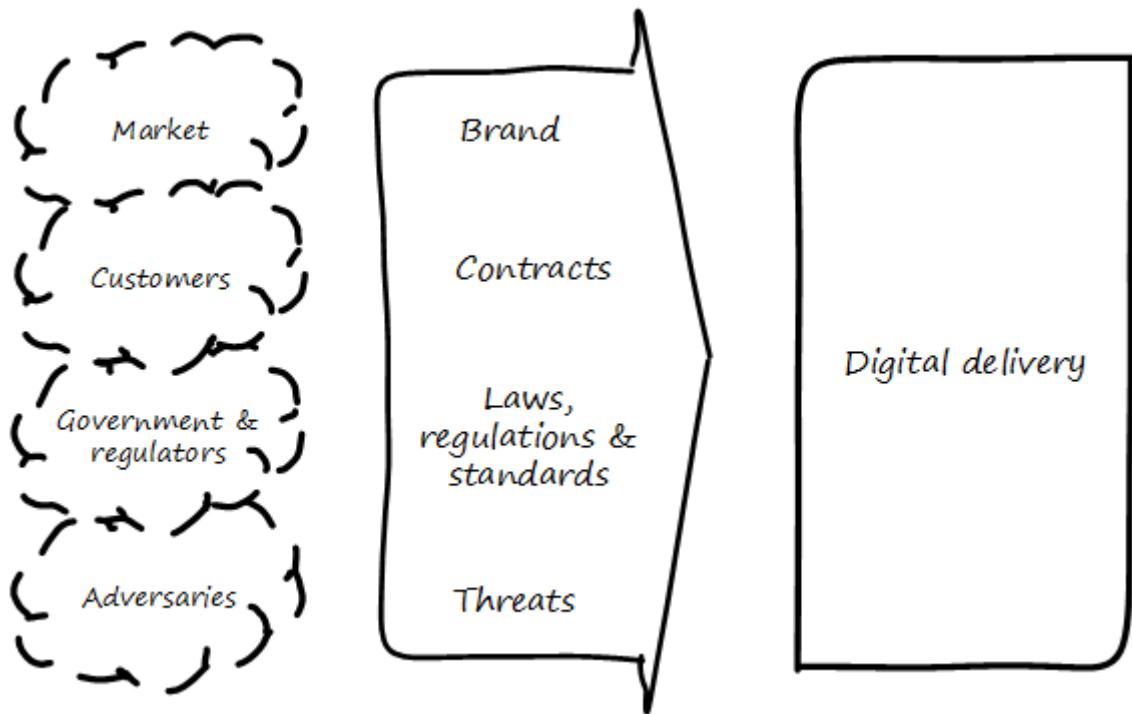
"Governors" have been part of mechanical systems for centuries. These mechanisms have the effect of automatically controlling a system so that it (for example) operates at the desired revolutions per minute. Without governors, steam engines tended to blow up, or go out; applying devices such as the **centrifugal governor** to regulate them was an important step in the development of steam power. Importantly, such devices operated to control the process from variation on either side, whether too fast or too slow. They did not operate merely as brakes.

So, if you find yourself coping with arbitrary and bureaucratic "governance" processes, it might be good to remember the origins of the term. There is more to governance than just slowing a system down.

Analyzing governance

Governance context

Enough theory. Governance is also practical concern for you because at your scale, you have a complex set of environmental forces to cope with:



You started with a focus on the customer, and the market they represented. Sooner or later, you encountered regulators and adversaries: competitors and cybercriminals.

Mark Burgess and Promise Theory

Promise Theory is . . . an engineering framework for coping with uncertainty in information systems.

— Mark Burgess, Promise Theory

We use the term "promise" here in a manner intended to be roughly consistent with Promise Theory. Promise Theory is a way of reasoning about relationships and systems developed by Mark Burgess. [\[Burgess2015\]](#)

Promise Theory is a sophisticated, mathematically-founded framework for reasoning about intent, outcome, control, and related topics of interest to digital governance. It is not possible to discuss it in depth here, but one basic heuristic we will follow is that a promise is better than a command, because "A promise expresses intent about the end point, or ultimate outcome, instead of indicating what to do at the starting point." We have explored similar themes in our chapter section on [culture](#).

These external parties intersect with your reality via various channels:

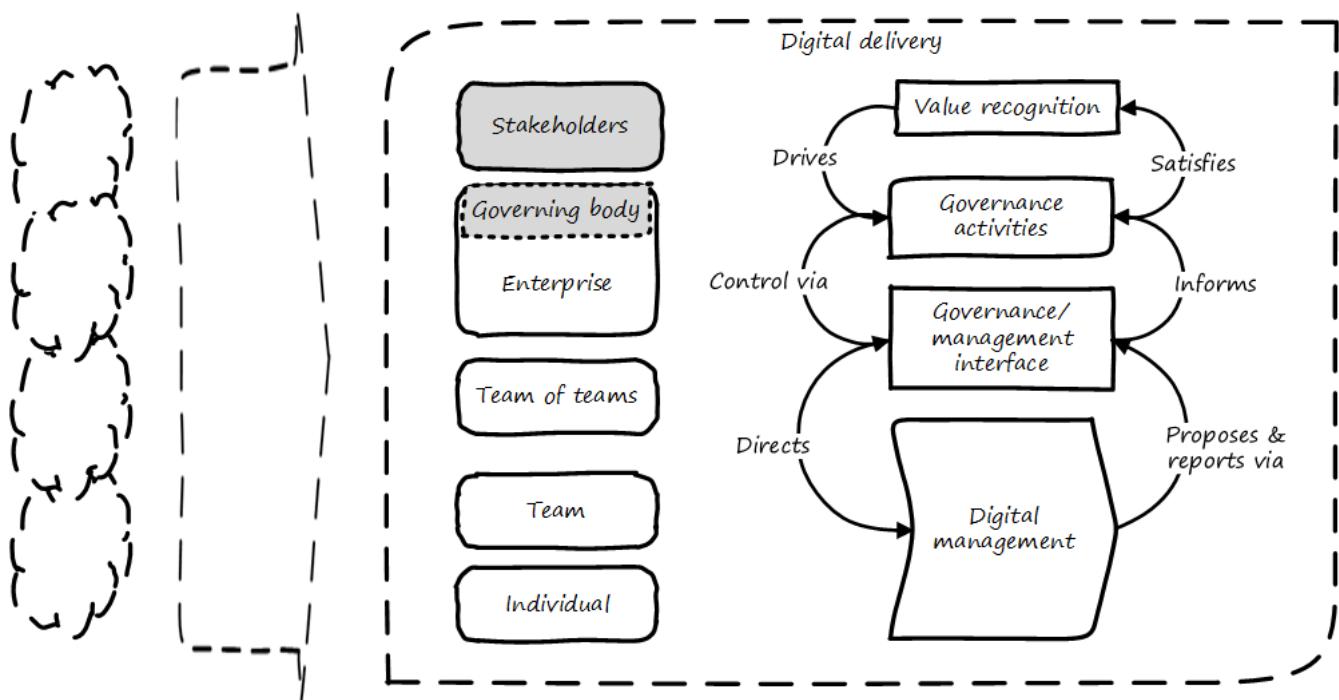
- Your brand, which represents a sort of general promise to the market (see [Sussna2015], p.16)
- Contracts, which represent more specific promises to suppliers and customers
- Laws, regulations, and standards, which can be seen as promises you must make and keep in order to function in civil society, or in order to obtain certain contracts.
- Threats, which may be of various kinds:
 - legal
 - operational
 - intentional
 - unintentional
 - illegal
 - environmental

We will return to the role of external forces in our discussion of assurance. For now, we will turn to how digital governance, within an overall system of digital delivery, reflects our emergence model.

credit Sussna for "brand as promise" <https://medium.com/@jeffsussna/brands-as-promise-marks-d43dc2efaf21#.24pc4smy5>

Governance and the emergence model

In terms of our **emergence model**, the most important distinction between a "**team of teams**" and an "**enterprise**" is the existence of **formalized** organizational governance.



In the above diagram, formalized governance is represented by the establishment of a **governing body**, responsive to some **stakeholders** who seek to recognize value from the organization or

"entity"—in this case, a digital delivery organization.

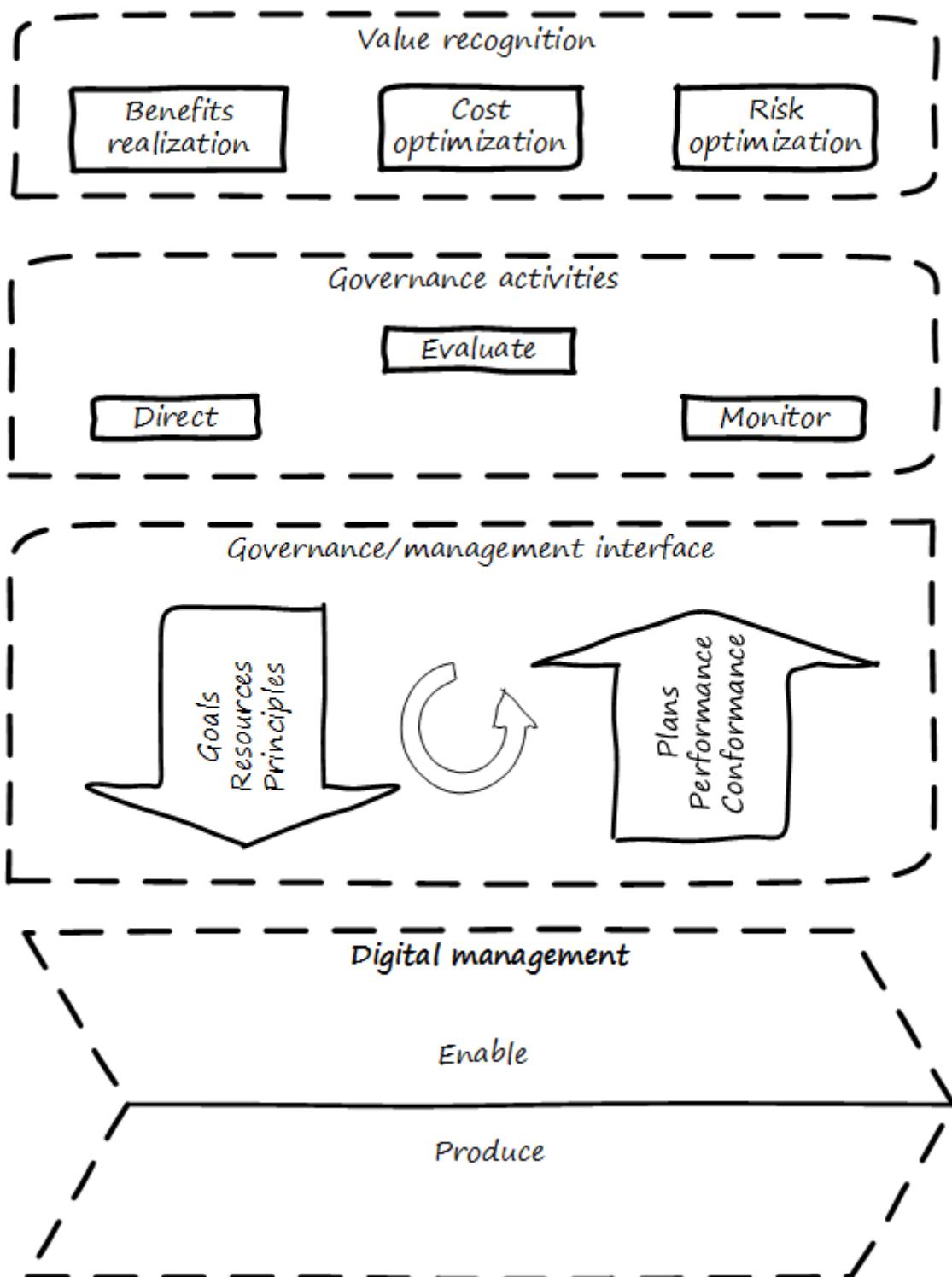
Corporate governance is a broad and deep topic, essential to the functioning of society and its organized participants. These include for-profit, non-profit, and even governmental organizations. Any legally organized entity of significant scope has governance needs.

One well known structure for organizational governance is seen in the regulated, publicly owned company (such as those listed on stock exchanges). In this model, shareholders elect a governing body (usually termed the Board of Directors), and this group provides the essential direction for the enterprise as a whole.

However, organizational governance takes other forms. Public institutions of higher education may have a Board of Regents or Board of Governors, perhaps appointed by elected officials. Nonprofits and incorporated private companies still require some form of governance, as well. One of the less well-known but very influential forms of governance is the venture capital portfolio approach, very different from a public, mission-driven company. We will talk more about this in the digital governance section.

These are well known topics in law, finance, and social organization, and there are many sources you can turn to if you have further interest. If you are taking any courses on Finance or Accounting, you will likely cover governance objectives and processes.

Here is a more detailed visual representation of the relationship of governance and management in a digital context:



synthesized from various sources including ISO 38500 and COBIT

Reading from the top down:

Value recognition is the fundamental objective of the stakeholder. We discussed in Chapter 4 the value objectives of **effectiveness**, **efficiency**, and **risk** (aka top line, bottom line, and risk). These are useful final targets for impact mapping, to demonstrate that lower level perhaps more "technical" product capabilities do ultimately contribute to organization outcomes.

NOTE

The term "value recognition" as the stakeholder goal is chosen over "value creation" as "creation" requires the entire system. Stakeholders do not "create" without the assistance of management, delivery teams, and the individual.

Here, we see them from the stakeholder perspective of

- Benefits realization
- Cost optimization
- Risk optimization

(Adapted from [\[ISACA2012a\]](#), p. 23)

Both ISO 38500 [\[ISO2008\]](#) as well as COBIT [\[ISACA2012a\]](#) specify that the fundamental **governance activities** of governance are:

- Direct
- Evaluate
- Monitor

Evaluation is the analysis of current state, including current proposals and plans. **Directing** is the establishment of organizational intent as well as the authorization of resources. **Monitoring** is the ongoing attention to organizational status, as an input to evaluation and direction.

Direct, Evaluate, and Monitor may also be ordered as Evaluate, Direct, and Monitor. These are highly general concepts that in reality are performed simultaneously, not as any sort of strict sequence.

The **governance/management interface** is an essential component. The information flows across this interface are typically some form of the following:

From the governing side

- Goals (e.g. product and go-to-market strategies)
- Resource authorizations (e.g. organizational budget approvals)
- Principles and policies (e.g. personnel and expense policies)

From the governed side

- Plans & proposals (at a high level, e.g. budget requests)
- Performance reports (e.g. sales figures)
- Conformance/compliance indicators (e.g. via audit and assurance)

Notice also the circular arrow at the center of the Governance/Management interface. Governance is not a one-way street. Its principles may be stable, but approaches, tools, practices, processes, and so forth (what we will discuss below as "enablers," in COBIT terminology) are variable, and require ongoing evolution.

We often hear of "bureaucratic" governance processes. But the problem may not be "governance" per se. It is more often the failure to correctly manage the governance/management interface. Of course, if the board is micro-managing, demanding many different kinds of information and intervening in operations, then governance and its management response is all much the same thing. In reality, however, burdensome organizational "governance" processes may be an overdone, bottom-up management response to **perceived** Board-level mandates.

Or they may be point-in-time requirements no longer needed. The policies of 1960 are unsuited to the realities of 2020. But if policies are always dictated top-down, they may not be promptly corrected or retired when no longer applicable. Hence, the scope and approach of governance in terms of its enablers must always be a topic of ongoing, iterative negotiation between the governed and the governing.

Finally the lowermost **digital delivery** chevron—aka [value chain](#), represents most of what we have discussed in Sections I, II, and III:

- the individual working to create value using digital infrastructure and lifecycle pipelines
- the team collaborating to discover and deliver valuable digital products
- the team of teams coordinating to deliver higher-order value while balancing effectiveness with efficiency and consistency

Ultimately, governance is about managing results and risk. It's about objectives and outcomes. It's about "what," not "how." In terms of practical usage, it is advisable to limit the "governance" domain— including use of the term—to a narrow scope of board or director-level concerns, and the existence of certain capabilities, including:

- organizational policy management
- external and internal assurance and audit
- risk management, including security aspects
- compliance

We turn to a useful concept for the implementation of digital governance—the concept of enablers.

Enablers

An introduction to enablers

Enablers are factors that, individually and collectively, influence whether something will work — in this case, governance and management over enterprise IT.

— Cobit 5 Framework, ISACA

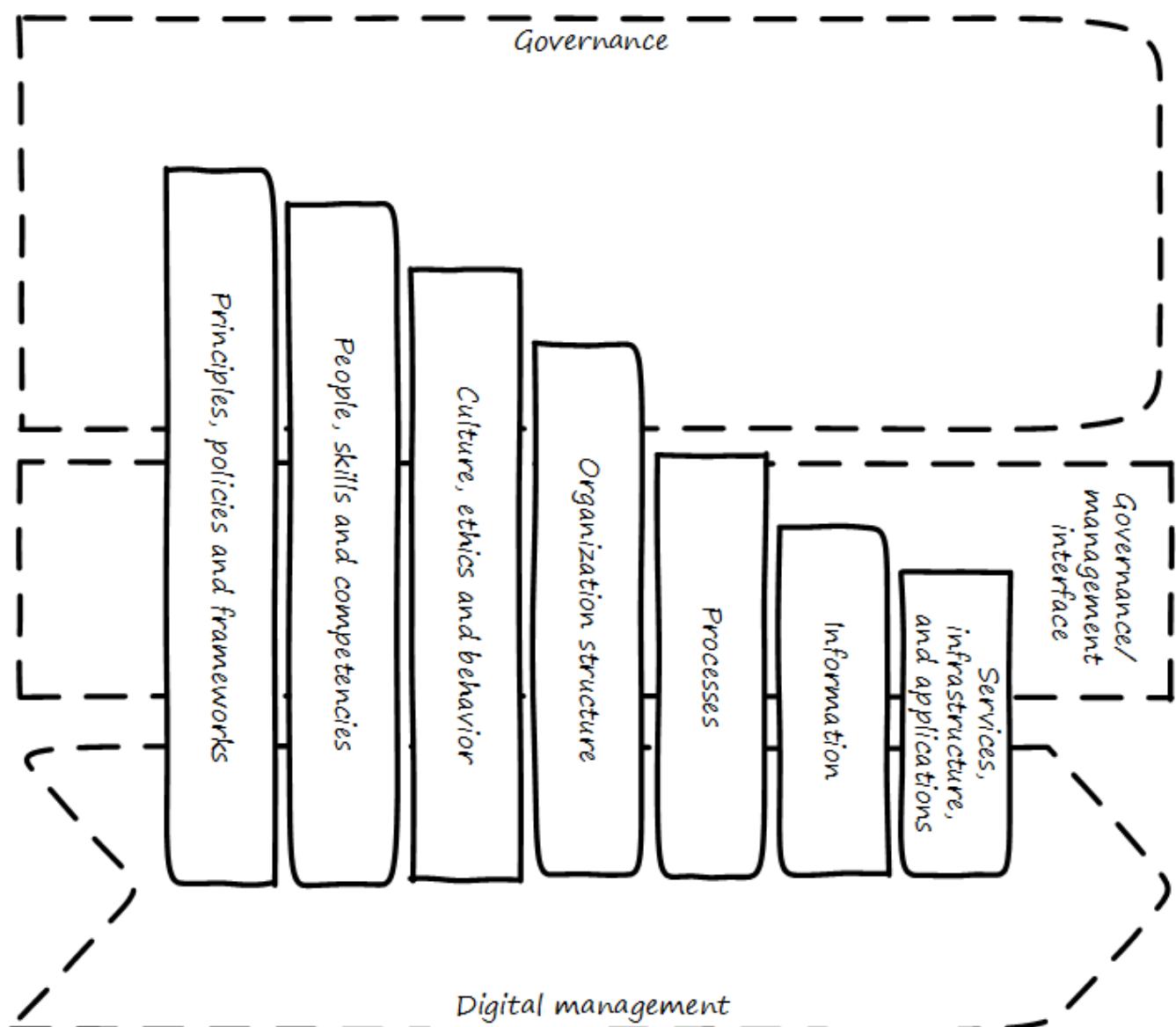
As we explore the governance/management interface further, we encounter the COBIT concept of

enablers [ISACA2012a].

Enablers are the fundamental components of any purposeful organization. COBIT has a detailed structure that positions enablers in a broader context of stakeholder objectives, enterprise and IT goals, and various quality criteria. One easy to understand example of a governance-level enabler concern is when processes serve as risk controls; we will discuss this further in the next chapter section.

COBIT's 7 enablers are:

- Principles, Policies, and Frameworks
- Processes
- Organizational Structures
- Culture, Ethics and Behavior
- Information
- Services, Infrastructure, and Applications
- People, Skills, and Competencies



The varying lengths of the enablers is deliberate. The further upward the bar extends, the more they are direct concerns for governance.

All of the enablers are discussed elsewhere in the book.

Enabler	Covered here
Principles, Policies, and Frameworks	Principles & policies covered in this chapter. Frameworks covered in Chapter 8 (PMBOK and Agile frameworks), Chapter 9 (CMMI , ITIL , COBIT , TOGAF), Chapter 11 (DMBOK), Chapter 12 (further TOGAF).
Processes	Chapter 9
Organizational Structures	Chapter 7
Culture, Ethics and Behavior	Chapter 7
Information	Chapter 11
Services, Infrastructure, and Applications	Section I
People, Skills, and Competencies	Chapter 8

Here, we are concerned with their aspect as presented to the governance interface; here are some notes.

Principles, Policies, and Frameworks

Principles are the most general statement of organizational vision and values. Policies will be discussed in detail in the next section. In general they are binding organization mandates or regulations, sometimes grounded in external laws. Frameworks were defined in [Chapter 8](#). We discuss all of these further in this chapter section.

NOTE

Some companies need to institute formal policies quite early. Even a startup may need written policies, if it is concerned with regulations such as HIPAA. However, this may be done on an ad-hoc basis, perhaps outsourced to a consultant. (A startup cannot afford a dedicated VP of Policy and Compliance.) This topic is covered in detail in this section because at enterprise scale, ongoing policy management and compliance must be formalized. Recall that [\[formalization\]](#) is the basis of our emergence model.

People, Skills, and Competencies

People and their skills and competencies (covered in [Chapter 8](#)) are an enabler upon which all the other enablers rest. "People are our #1 asset" may seem to be a cliche, but it is ultimately true. Formal approaches to understanding and managing this base of skills are therefore needed. A basic "Human Resources" capability is a start, but the most sophisticated and ambitious institute formal organizational learning capabilities to ensure that talent remains a critical focus.

Culture, Ethics and Behavior

Culture, ethics, and behavior as an enabler can both drive revenue as well as risk and cost. Recent research by Michael Houseman and Dylan Minor suggests that while the benefit from hiring a highly qualified "superstar" worker at most is \$5,303, the cost of hiring a "toxic" worker (one destructive of morale and team norms) averages \$12,489 - certainly a risk to consider [Housman2015]. Culture is discussed in [Chapter 7](#).

Organizational Structures

We discussed basic organizational structure in [Chapter 7](#). However, governance also may make use of some of the scaling approaches discussed in Chapter 8. Cross-organization techniques (similar to those discussed in [Chapter 8](#)) are frequently used in governance (e.g. cross-organizational coordinating committees, such as an enterprise security council).

Processes

Process was defined in [Chapter 9](#). We will discuss enablers as controls in the upcoming chapter section on risk management. A control is a role that an enabler may play. Processes are the primary form of enabler used in this sense.

Information

Information is a general term; in the sense of an enabler, it is based on data in its various forms, with overlays of concepts (such as syntax and semantics) that transform raw "data" into a resource that is useful and valuable for given purposes. From a governance perspective, information carries governance direction to the governed system, and the feedback monitoring also is transmitted as information. Information resource management and related topics such as data governance and data quality are covered in [Chapter 11](#); it is helpful to understand governance at an overall level before going into these more specific domains.

Services, Infrastructure, and Applications

Services, infrastructure, and applications of course are the critical foundation of digital value. These fundamental topics were covered in [Section I](#). In the sense of enablers, they have a recursive or self-reflexive quality. Digital technology automates business objectives; at scale, a digital pipeline becomes a nontrivial business concern in and of itself, requiring considerable automation [[Betz2011a](#)], [[Open2015](#)]. Applications that serve as digital governance enablers might include:

- Source control
- Build management
- Package management
- Deployment and configuration management
- Monitoring
- Portfolio management

Policy management

How policy begins

Your company was incorporated long ago, but the "board" was always a bit of a joke. The three people who started the company were the directors of record, and they would have an annual "meeting" at the local bar where enough paperwork would be done to satisfy the company lawyer.

Your company did well, and accumulated enough cash to purchase another company, run in much the same way. The people who owned the company being acquired were good, and your company didn't want to lose them, so in addition to senior management positions, they were offered equity—a share of ownership in the new combined firm.

This raised the topic, "how is the new firm directed?" One of the incoming shareholders wanted a seat on the "board," even though neither company had done much with board-level governance.

The lawyer and accountant hired to assist with the merger also recommended that as part of the acquisition, a formal audit be conducted of both firms (which had never been done).

This audit came back generally clean, but shone a light on differences in how the companies had operated, and unearthed some irregularities.

For example, your company had started to purchase phones for all employees, while the acquired company was pure BYOD (Bring Your Own Device). One company had corporate credit cards, while the other was requiring people to carry their own expenses for reimbursement. One company had an informal "understanding" that first class travel was OK for Asian trips at least, while the other didn't, but neither had written anything down. And so on.

The lawyer said, "I think you need some policies," and everybody groaned. One person said, "I just read about Nordstrom. All they say is "Use Good Judgment." Why do we need anything more?"

The lawyer said, "Um, that's an urban legend. The actual Nordstrom Code of Business Conduct and Ethics, while it starts off with that, runs about 8,000 words and covers a variety of topics such as handling customer information, using technology, social media, and so forth."

And the new CFO said, "Look, I get that we want to stay agile, and keep our informal culture. I'm no fan of policy for the sake of policy. But I need those policies to keep **my** staff costs down. Two different expense approaches doesn't add **any** value to us, and that's only one of twenty issues we've uncovered here. 'Do the right thing' doesn't cut it. We've got to have some means for establishing a baseline with new employees, someplace people can turn to when they don't know what the expectation is."

The HR director chimed in. "If we don't document our official position on things like harassment we are going to have problems. We could fire someone who has done something really bad, and they could sue us for wrongful termination. Or their victims could sue us for failing to prevent the issue. That could cost us real money." The lawyer nodded and the

company owners looked thoughtful.

Another person spoke up. "I came from a company that had a 500-page policy manual. It went down into way too much detail and was always out of date. No-one could find anything in it, and there would be stuff that was wrong because the revision process was broken."

The lawyer said, "You need to keep your policies light and on the general side (like Nordstrom), and cover more detailed topics elsewhere. For example, the exact approach on how to reimburse employee expenses probably doesn't belong in the policy manual. Of course, that means that somewhere you need to lay out how your principles inform your policies which are implemented by processes, procedures, guidelines, and so forth. Your actual employee handbook will probably be thirty or forty pages - sorry. You also should take advantage of your internal intranet and make sure people can find just the policy they need, with related guidance, instead of having to page through a huge document."

"Finally, you need to carefully distribute the authorship and revision control, especially for lower levels of the guidance (e.g. technical standards that can change quickly). This is both because the people most affected should have a stronger voice in the policy, and also because centralized policy groups become bottlenecks if they are doing all the work."

Another said, "This is all getting complicated."

"Yes, complexity is to some extent unavoidable as you move to this new scale. I'm a big fan of sunset dates on policies and supporting materials, so you are periodically questioning whether something is still needed. Of course, this drives demand for someone to analyze and update policies - please don't forget that."

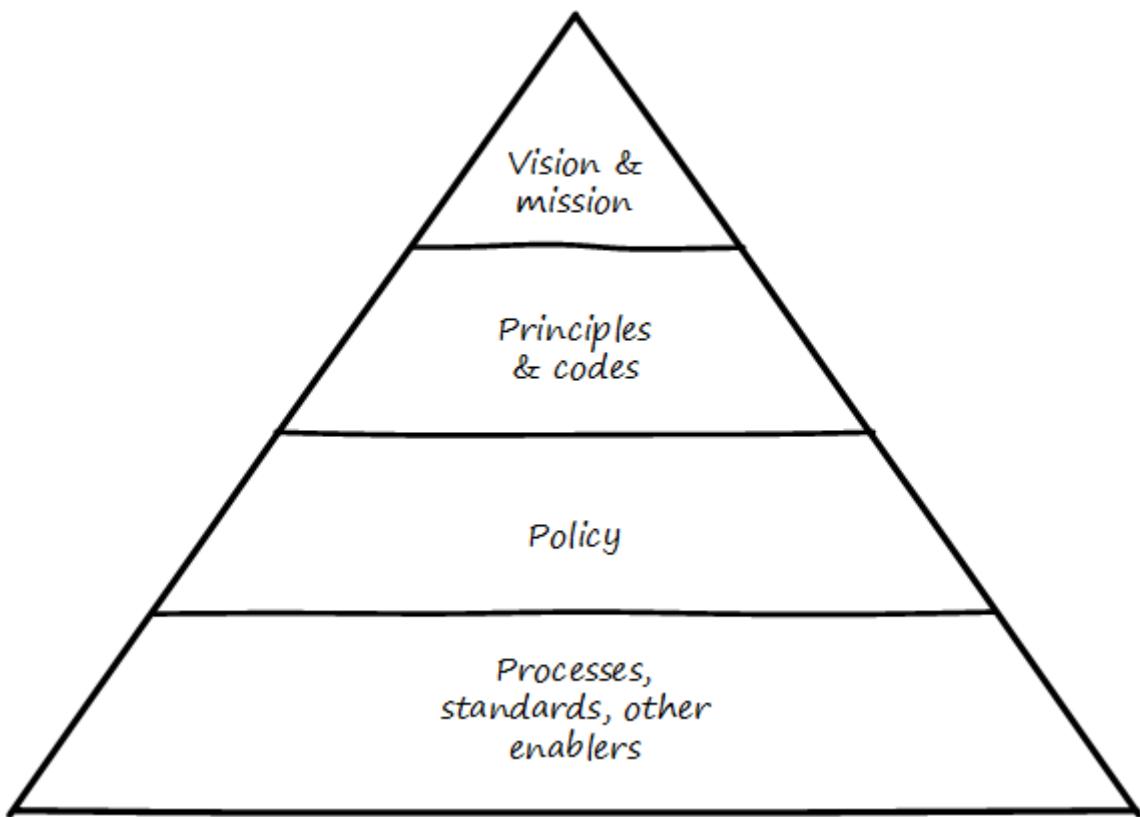
"Overall, you need to always keep your outcomes in mind, and continue to push as much decision making down to individuals as you can. COBIT recognizes that culture is one of the critical [enablers](#) for governance, and so 'use good judgment' is still a great place to start—IF you can hire people with good judgment, and continually reinforce them in using it."

see [\[Nordstrom2015\]](#), [\[Lucas2014\]](#)

Mission, principle, strategy, and policy

Carefully drafted and standardized policies and procedures save the company countless hours of management time. The consistent use and interpretation of such policies, in an evenhanded and fair manner, reduces management's concern about legal issues becoming legal problems.

— Michael Griffin, How To Write a Policy Manual



The above illustration shows one way to think about policy in the context of our overall governance objective of value recognition.

The organization's **Vision and mission** should be terse and high level, perhaps something that could fit on a business card. It should express the organization's reason for being in straightforward terms.

The **Principles and codes** should also be brief. ("Codes" can include codes of ethics or codes of conduct.) For example, Nordstrom's is about 8,000 words, perhaps about 10 pages.

Policies are more extensive. There are various kinds of policies:

In a non-IT example, a compliance policy might identify the Foreign Corrupt Practices act and make it clear that bribery of foreign officials is unacceptable. Similarly, an HR policy might spell out acceptable and unacceptable side jobs (e.g., someone in the banking industry might be forbidden from also being a mortgage broker on their own account).

Policies are often independently maintained documents, perhaps organized along lines similar to:

- Employment and HR policies
- Whistleblower policy (non-retaliation)
- Records retention
- Privacy
- Workplace guidelines
- Travel and expense
- Purchasing and vendor relationships

- Use of enterprise resources
- Information security
- Conflicts of interest
- Regulatory

(not a comprehensive list)

Policies, even though more detailed than codes of ethics/conduct, still should be written fairly broadly. In many organizations, they must be approved by the governing board. **They should therefore be independent of technology specifics.** An information security policy may state that the hardening guidelines must be followed, but the hardening guidelines (stipulating for example what services and open ports are allowable on Debian Linux) are **not** policy.

There may be various levels or classes of policy.

Finally, policies reference **Standards and processes** and other [enablers](#) as appropriate. This is the management level, where documentation is specific and actionable. Guidance here may include:

- Standards
- Baselines
- Guidelines
- Processes and procedures

These concepts may vary according to organization, and can become quite detailed.

Even more detail is seen in hardening guidelines. A behavioral baseline might be "Guests are expected to sign in and be accompanied when on the data center floor."

We will discuss technical baselines further in the chapter section on security, and also in our discussion of the technology product lifecycle in Chapter 12. See also Shon Harris' excellent CISSP Exam Guide [[Harris2013](#)] for much more detail on these topics.

The ideal end state is a policy that is completely traceable to various automation characteristics, such as approved "infrastructure as code" settings applied automatically by configuration management software (as discussed in "The DevOps Audit Toolkit," [[DeLucia2015](#)]-- more on this to come). However, there will always be organizational concerns that cannot be fully automated in such manners.

Policies (and their implementation as processes, standards, and the like) must be enforced. As Steve Schlarman notes "Policy without a corresponding compliance measurement and monitoring strategy will be looked at as unrealistic, ignored dogma." [[Schlarman2008](#)]

Finally, policies and their derivative guidance are developed, just like systems, via a lifecycle. They require some initial vision, and an understanding of what the requirements are. Again, Schlarman: "policy must define the why, what, who, where and how of the IT process" [[Schlarman2008](#)]. User stories have been used effectively to understand policy needs.

Finally, an important point to bear in mind:

Company policies can breed and multiply to a point where they can hinder innovation and risk-taking. Things can get out of hand as people generate policies to respond to one-time infractions or out of the ordinary situations [Griffin2016], p. 17.

It's advisable to institute sunset dates or some other mechanism that forces their periodic review, with the understanding that any such approach generates demand on the organization that must be funded. We will discuss this more in the chapter section on digital governance.

Standards, frameworks, methods, and the innovation cycle

We used the term "standards" above without fully defining it.

We have discussed a variety of industry influences throughout this book: PMBOK, ITIL, COBIT, Scrum, Kanban, ISO/IEC 38500 and so on. We need to clarify their roles and positioning further.

All of these can be considered various forms of "guidance" and as such are governance **enablers**. However, their origins, stakeholders, format, content, and usage vary greatly.

First, the term "**standard**" especially has multiple meanings. A "standard" in the policy sense may be a set of compulsory rules. Also, "standard" or "baseline" may refer to some intended or documented state the organization uses as a reference point. An example might be "we run Debian Linux 16.10 as a standard unless there is a compelling business reason to do otherwise."

This last usage shades into a third meaning of standard, normative standards such as are produced by the IEEE, IETF and ISO/IEC.

- ISO/IEC: International Standards Organization/International Electrotechnical Commission
- IETF: Internet Engineering Task Force
- IEEE: Institute of Electrical and Electronics Engineers

The International Standards Organization occupies a central place in this ecosystem. It possesses "general consultative status" with the United Nations, and has over 250 technical committees that develop the actual standards.

The IEEE standardizes such matters as wireless networking (e.g. WiFi). The IETF (Internet Engineering Task Force) standardizes lower level Internet protocols such as TCP/IP and HTTP. The W3C (World Wide Web Consortium) standardizes higher level Internet protocols such as HTML. Sometimes standards are first developed by a group such as the IEEE/IEC and then given further authority through publication by ISO/IEC.

The ISO/IEC in particular, in addition to its technical standards, also develops higher order management/"best practice" standards. One well known example of such an ISO standard is the ISO 9000 series on quality management.

There are crucial non-IT management standards as well, such as ... Some of these standards may have a great effect on the digital organization. We'll discuss this further in the chapter section on compliance.

Frameworks were discussed in [Chapter 9](#). Frameworks have two major meanings. First, computing language frameworks are created to make software development easier. Examples include Struts,

AngularJS, and many more. This is a highly volatile area of technology, with new frameworks appearing every year and older ones gradually losing favor.

In general we are not concerned with these kinds of specific frameworks in this book, except governing them as part of the [technology product lifecycle](#). We are concerned with "process" frameworks such as ITIL, PMBOK, COBIT, CMMI, and TOGAF. These frameworks are not "standards" in and of themselves. **However**, they often have corresponding ISO standards:

Framework	Standard
ITIL	ISO/IEC 20000
COBIT	ISO/IEC 38500
PMBOK	ISO/IEC 21500
CMMI	ISO/IEC 15504
TOGAF	ISO/IEC 42010

Frameworks tend to be lengthy and verbose. The ISO/IEC standards are brief by comparison, perhaps on average 10% of the corresponding framework.

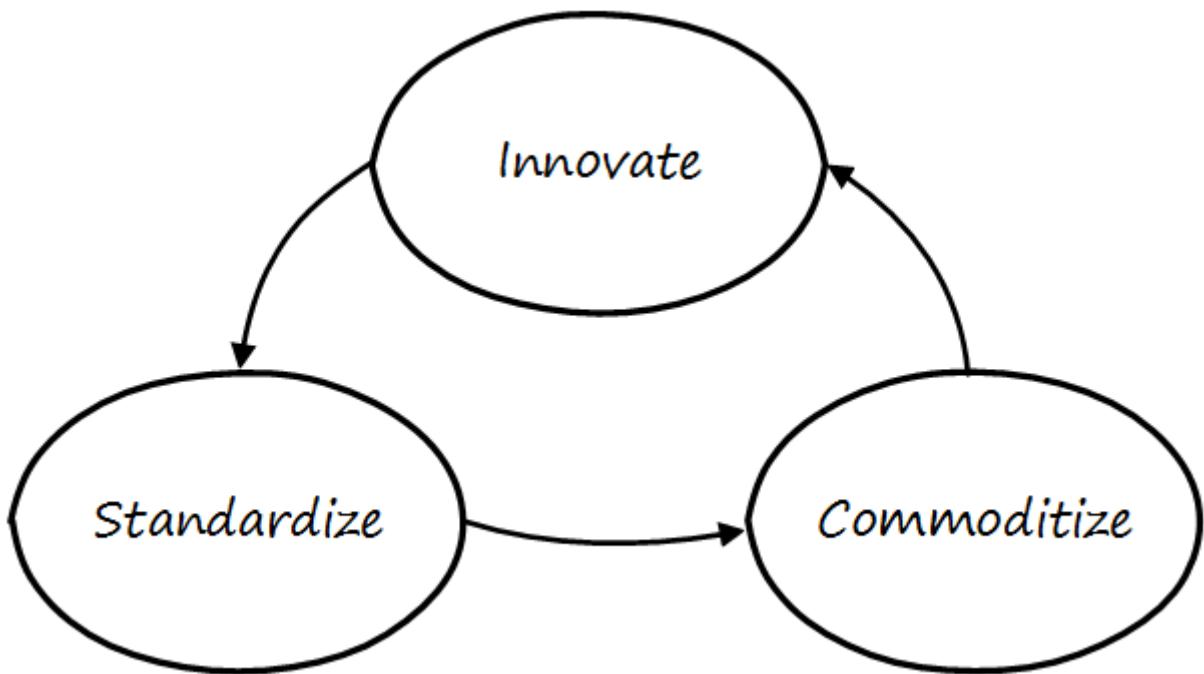
Methods (aka methodologies) in general are more action oriented and prescriptive. Scrum and XP are methods. It is at least arguable that PMBOK is a method as well as a framework.

NOTE

There is little industry consensus on some of these definitional issues and the student is advised to not be overly concerned about such abstract debates. If you need to comply with something to win a contract, it doesn't matter whether it's a "standard," "framework," "guidance," "method," or what have you.

Finally, there are terms that indicate technology cycles, movements, communities of interest, or cultural trends: Agile and DevOps being two of the most current and notable. These are neither frameworks, standards, nor methods. However, commercial interests often attempt to build frameworks and methods representing these organic trends. Examples include the Scaled Agile Framework, Disciplined Agile Delivery, and many others.

Ultimately, a standardization cycle can be seen:



Innovations produce value, but innovation presents change management challenges, such as cost and complexity. The natural response is to standardize for efficiency, and standardization taken to its end state results in commodification, where costs are optimized as far as possible and the remaining concern is managing the risk of the commodity (as either consumer or producer). While efficient, commoditized environments offer little competitive value, and so the innovation cycle starts again.

Note that the innovation cycle corresponds to the elements of [value recognition](#):

- Innovation corresponds to Benefits Realization
- Standardization corresponds to Cost Optimization
- Commoditization corresponds to Risk Optimization

Risk management

Risk management fundamentals

Risk is defined as the possibility that an event will occur and adversely affect the achievement of objectives.

— Committee of Sponsoring Organizations of the Treadway Commission, Internal Control - Integrated Framework

Risk is a fundamental concern of governance. Management (as we have defined it in this chapter section) may focus on effectiveness and efficiency well enough, but too often disregards risk.

As we noted above, the shop manager may have incentives to maximize income, but usually does not stand to lose their life savings. The owner, however, does. Fire, theft, disaster - without risk management, the owner does not sleep well.

For this reason, risk management is a large element of governance, as indicated by the popular GRC acronym: Governance, Risk, and Compliance.

Defining "Risk"

The definition of "risk" is surprisingly controversial. The ISO 31000 standard [ISO2009] and the Project Management Institute's PMBOK [PMI2013] both define risk as including positive outcomes (benefits). This definition has been strongly criticized by (among others) Douglas Hubbard in *The Failure of Risk Management* [Hubbard2009]. Hubbard points out that traditionally, risk has meant the chance and consequences of loss.

As this is an overview text, we will use the more pragmatic, historical definition. Practically speaking, Operational Risk Management as a function focuses on loss. The possibility ("risk") of benefits is eagerly sought by the organization as a whole and does not need "management" by a dedicated function.

"Loss," however, can also equate to "failure to achieve anticipated gains." This form of risk applies (for example) to product and project investments.

add risk v uncertainty Hubbard

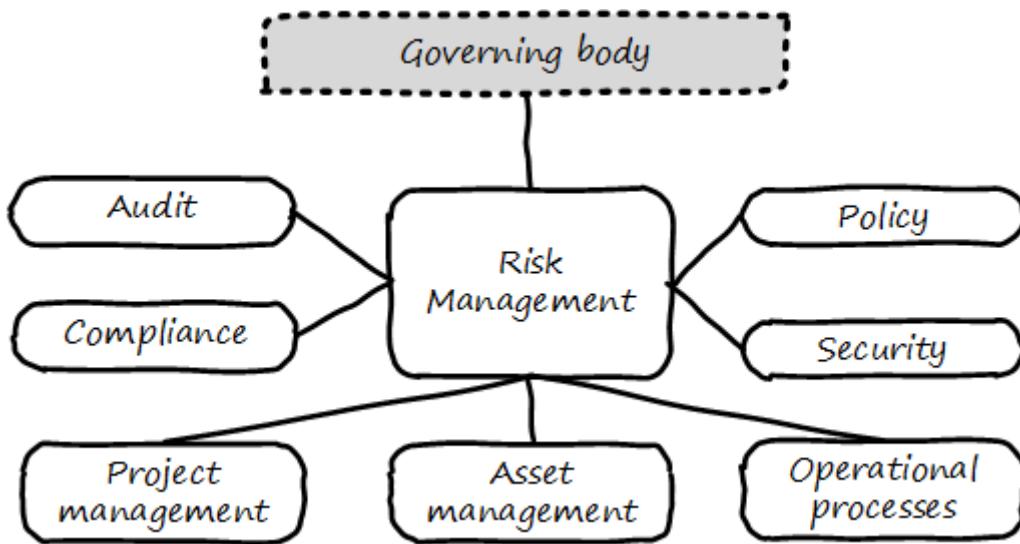
Risk management can be seen as both a **function and a process**. As a function, it may be managed by a dedicated organization (perhaps called Enterprise Risk Management or Operational Risk Management.) As a process, it conducts the following activities:

- Identifying risks
- Assessing and prioritizing them
- Coordinating effective responses to risks
- Ongoing monitoring and reporting of risk management

Risk impacts some asset. Examples in the digital and IT context would include:

- Operational IT systems
- Hardware (e.g. computers) and facilities (e.g. data centers)
- Information (customer or patient records)

It is commonly said that organizations have an "appetite" for risk ([ISACA2013], p. 79), in terms of the amount of risk the organization is willing to accept. This is a strategic decision, usually reserved for organizational governance.



Risk management typically has strong relationships with the following organizational capabilities:

- Enterprise governance (e.g. board-level committees)
- Security
- Compliance
- Audit
- Policy management

For example, security requires risk assessment as a input, so that security resources focus on the correct priorities.

Risk additionally may interact with:

- Project management
- Asset management
- Processes such as Change Management

and other digital activities.

More detail on core risk management activities follows, largely adopted from the COBIT for Risk publication ([ISACA2013](#)).

Risk identification

There are a wide variety of potential risks, and many accounts and anecdotes constantly circulating. It is critical that risk identification start with a firm understanding of the organization's objectives and context.

Risk identification can occur both in a "top down" and "bottom up" manner. Industry guidance can assist the operational risk management function in identifying typical risks. For example, the COBIT for Risk publication includes a useful 8 page "Generic Risk Scenarios" section ([ISACA2013](#), pp. 67-74) identifying risks such as

- "Wrong programmes are selected for implementation and are misaligned with corporate

strategy and priorities"

- "There is an earthquake."
- "Sensitive data is lost/disclosed through logical attacks."

These are only three of dozens of scenarios. Risks of course extend over a wide variety of areas:

- Investment
- Sourcing
- Operations
- Availability
- Continuity
- Security

and so forth. The same guidance also strongly cautions against over-reliance on these generic scenarios.

Risk assessment

Risk management has a variety of concepts and techniques both qualitative and quantitative. Risk is often assumed to be the product of probability * impact. For example, if the chance of a fire in a facility is 5% over a given year, and the damage of the fire is estimated at \$100,000, the annual risk is \$5,000. An enterprise risk management function may attempt to quantify all such risks into an overall portfolio.

Where quantitative approaches are perceived to be difficult, risk may be assessed using simple ordinal scales (e.g. 1-5, where 1 is low risk and 5 is high risk.) COBIT for Risk expresses concern regarding "The use of ordinal scales for expressing risk in different categories, and the mathematical difficulties or dangers of using these numbers to do any sort of calculation." [\[ISACA2013\]](#), p. 75. Such approaches are criticized by Doug Hubbard in *The Failure of Risk Management* as misleading and potentially more harmful than not managing risk at all [\[Hubbard2009\]](#).

Hubbard instead suggests that quantitative techniques such as Monte Carlo analysis are rarely infeasible, and recommends their application instead of subjective scales.

The enterprise can also consider evaluating scenarios that have a chance of occurring simultaneously. This is frequently referred to as 'stress' testing.

Risk response

He who fights and runs away, lives to fight another day.

— Menander, 342 BC - 291 BC

Risk response includes several approaches:

- Avoidance

- Acceptance
- Transference
- Mitigation

Avoidance means ending the activities or conditions causing the risk; e.g., not engaging in a given initiative or moving operations away from risk factors.

Acceptance means no action is taken. Typically, such "acceptance" must reside with an executive.

Transference means that some sharing arrangement, usually involving financial consideration, is established. Common transfer mechanisms include outsourcing and insurance. (Recall our discussion of Agile approaches to [contract management](#) and risk sharing.)

Mitigation means that some compensating mechanism — one or more "controls" is established. This topic is covered in the next section, and comprises the remainder of the material on risk management.

The above discussion was largely derived from [\[ISACA2013\]](#).

Controls

The term *control objective* is no longer a mainstream term used in COBIT 5, and the word *control* is used only rarely. Instead, COBIT 5 uses the concepts of process practices and process activities.

— ISACA, COBIT 5 for Assurance

The term "control" is problematic.

It has distasteful connotations to those who casually encounter it, evoking images of "command and control" management, or "controlling" personalities. COBIT, which once stood for Control Objectives for IT, now deprecates the term control (see the above quote). Yet it retains a prominent role in many discussions of enterprise governance and risk management, as we saw at the start of this chapter in the discussion of [COSO's general concept of control](#).

And (as discussed in our coverage of [Scrum's origins](#)) it is a technical term of art in systems engineering. As such it represents principles essential to understanding large scale digital organizations.

In this section, we are concerned with controls in a narrower sense, as risk mitigators.

As noted above, ISACA replaced the term "controls" with process practices & activities, which are specific examples of enablers. As controls, enablers such as policies, procedures, organizational structures, and the rest are used and intended to ensure that:

- investments achieve their intended outcomes;
- resources are used responsibly, and protected from fraud, theft, abuse, waste, and

mismanagement;

- laws and regulations are adhered to; and
- timely and reliable information is employed for decision making.

NOTE

You will likely encounter the term control as "testing enablers" is not how security personnel and auditors talk.

But what are examples of "controls"? Take a risk, such as the risk of a service (e.g. ecommerce Web site) outage resulting in loss of critical revenues.

There are a number of ways we might attempt to mitigate this risk:

- Configuration management (a **preventative** control)
- Effective monitoring of system alerts (a **detective** control)
- Documented operational responses to detected issues (a **corrective** control)
- Clear recovery protocols that are practiced and well understood (a **recovery** control)
- System redundancy of key components where appropriate (a **compensating** control)

and so forth. Another kind of control appropriate to other risks is **deterrent** (e.g. an armed guard at a bank).

Other types of frequently seen controls include:

- Separation of duties
- Audit trails
- Documentation
- Standards and guidelines

A control type such as "Separation of duties" is very general and might be specified by activity type, e.g.

- Purchasing
- System development and release
- Sales revenue recognition

Each of these would require distinct approaches to separation of duties. Some of this may be explicitly defined; if there is no policy or control specific to a given activity, an auditor may identify this as a deficiency.

Policies and processes in their aspect as controls are often what auditors test. In the case of the web site above, an auditor might test the configuration management approach, the operational processes, inspect the system redundancy, and so forth. And risk management would maintain an ongoing interest in the system in between audits.

As with most topics in this book, risk management (in and of itself, as well as applied to IT and digital) is an extensive and complex domain, and this discussion was necessarily brief. The student

is referred to the readings at the end of the chapter for further information.

Business continuity

Business continuity is an applied domain of IT risk, like security. Continuity is concerned with large scale disruptions to organizational operations, such as:

- Floods
- Earthquakes
- Tornadoes
- Terrorism
- Hurricanes
- Industrial catastrophes (e.g. large scale chemical spills)

A distinction is commonly made between:

- Business Continuity Planning
- Disaster Recovery

Disaster recovery is more tactical, including the specific actions taken during the disaster to mitigate damage and restore operations.

Continuity planning takes a longer term view of matters such as long-term availability of replacement space and computing capacity.

There are a variety of standards covering business continuity planning, including:

- NIST Special Publication 800-34
- ISO/IEC 27031:2011
- ISO 22301

In general, continuity planning starts with understanding business impact of various disaster scenarios and developing plans to counter them. Traditional guidance suggests that this be done in a centralized fashion; however, large, centralized efforts of this nature tend to struggle for funding.

While automation alone cannot solve problems such as "where do we put the people if our main call center is destroyed," it can help considerably in terms of recovering from disasters. If a company has been diligent in applying **Infrastructure as Code** techniques, and loses its data center, it can theoretically re-establish its system configurations readily, which can otherwise be a very challenging process, especially under time constraints. (Data still needs to have been backed up to multiple locations.)

Compliance

Compliance is a very general term meaning conformity or adherence to

- laws
- regulations
- policies
- contracts
- standards

and the like. Corporate compliance functions may first be attentive to legal and regulatory compliance, but the other forms of compliance are matters of concern as well.

A corporate compliance office may be responsible for the maintenance of organizational policies and related training and education, perhaps in partnership with the HR department. They also may monitor and report on the state of organizational compliance. Compliance offices may also be responsible for codes of ethics. Finally, they may manage channels for anonymous reporting of ethics and policy violations by whistleblowers (individuals who become aware of and wish to report violations while receiving guarantees of protection from retaliation).

Compliance uses techniques similar to risk management, and in fact non-compliance can be managed as a form of risk, and prioritized and handled much the same way.

However, compliance is an information problem as well as a risk problem. There is an ongoing stream of regulations to track, which keeps compliance professionals very busy. In the U.S. alone, these include:

- HIPAA
- SOX
- FERPA
- PCI DSS
- GLBA PII (NIST)

Some of these regulations specifically call for policy management, and therefore companies that are subject to them may need to institute formal governance earlier than other companies, in terms of the emergence model. Many of them provide penalties for the mis-management of data, which we will discuss further in the next chapter section and in Chapter 11.

Compliance also includes compliance with the courts (e.g. civil and criminal actions). This will be discussed in the Chapter 11 section on cyberlaw.

Assurance and audit

Assurance

Trust, but verify.

— Russian proverb

Assurance is a broad term. In this book, it is associated with governance. It represents a form of additional confirmation that management is performing its tasks adequately.

Go back to the [example](#) that started the chapter, of the shop owner hiring a manager. Suppose that this relationship has continued for some time, and while things seem to be working OK, the owner has doubts about the arrangement.

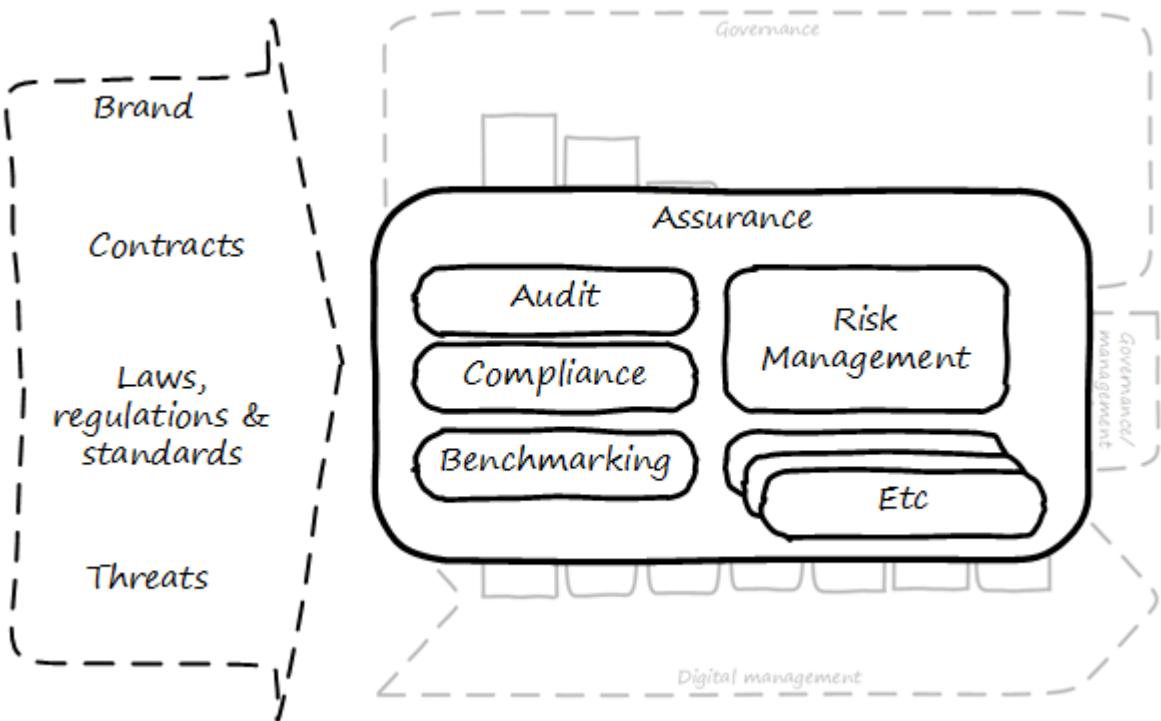
Over time, things have gone well enough that the owner does not worry about the shop being opened on time, having sufficient stock, or paying suppliers. But are any number of doubts the owner might retain:

- Is money being accounted for honestly and accurately?
- Is the shop clean? Is it following local regulations? For example, fire, health and safety codes?
- If the manager selects a new supplier, are they trustworthy? Or is the shop at risk of selling counterfeit or tainted merchandise?
- Are the shop's prices competitive? Is it still well regarded in the community? Or has its reputation declined with the new manager?
- Is the shop protected from theft and disaster?

These kinds of concerns remain with the owner, by and large, even with a reliable and trustworthy manager. If not handled correctly, the owner's entire investment is at risk. The manager may only have a salary (and perhaps a profit share) to worry about, but if the shop is closed due to violations, or lawsuit, or lost to a fire, the owner's entire life investment may be lost.

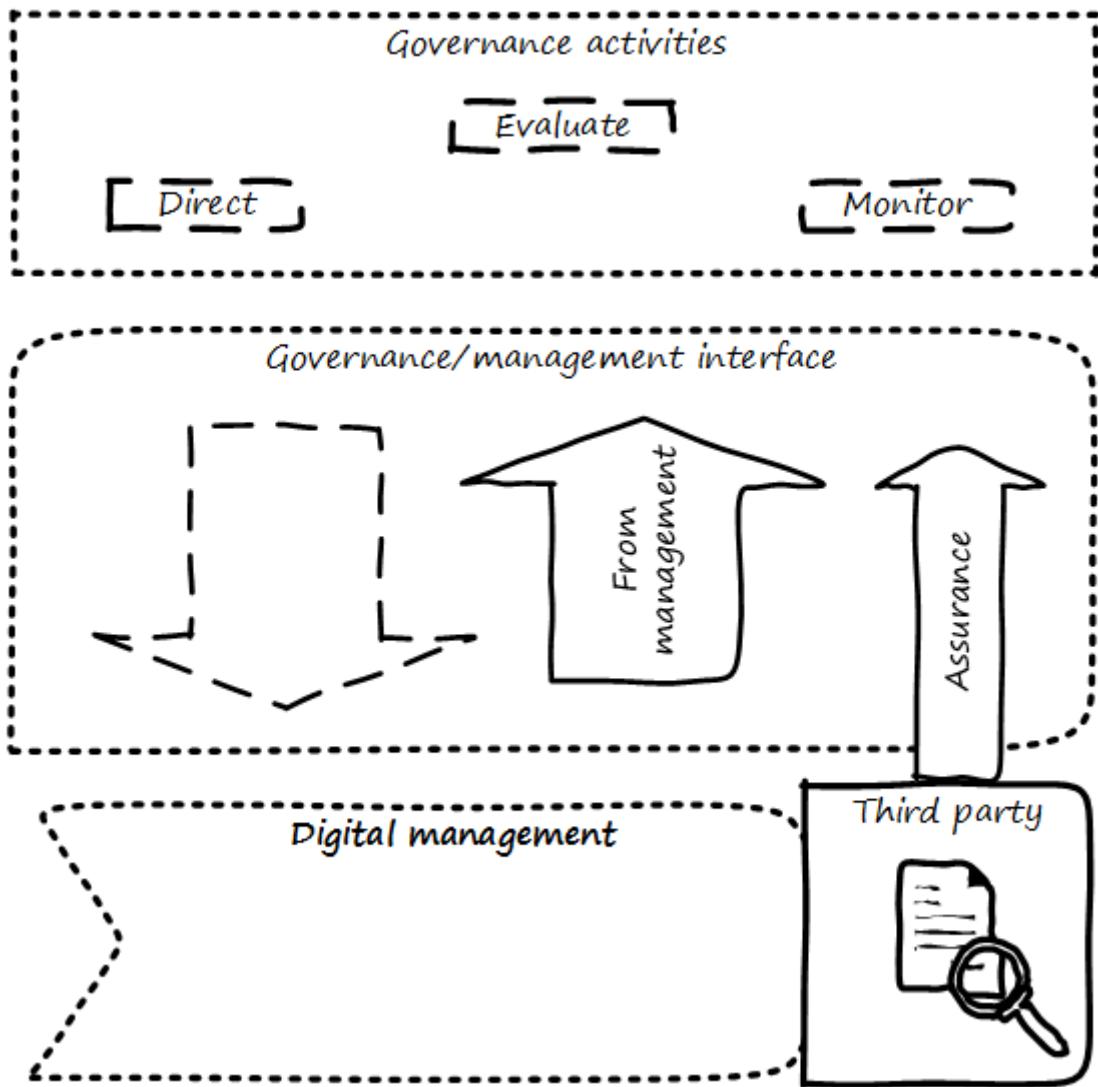
These concerns give rise to the general concept of assurance, which applies to digital business just as it does to small retail shops.

The following diagram, derived from previous illustrations, shows how this book views assurance: as a set of practices overlaid across the enablers, and in particular concerned with external forces.



As ISACA stipulates,

The IS audit and assurance function shall be independent of the area or activity being reviewed to permit objective completion of the audit and assurance engagement. [ISACA2014], p.9. Assurance can be seen as an external, additional mechanism of control and feedback. This independent, out-of-band aspect is essential to the concept of assurance.



Assurance is like out-of-band management

In terms of the governance-management interface, assurance is fundamentally distinct from the information provided by management and must travel through distinct communication channels. This is why auditors (for example) forward their reports directly to the audit committee, and do not route them through the executives who have been audited.

Technologists, especially those with a background in networking, may have heard of the concept of "out of band control." With out of band management or control of IT resources, the channel over which management commands travel is distinct from the channel over which the system provides its services.

This channel separation is done to increase security, confidence, and reliability, and is analogous to assurance.

Three party foundation

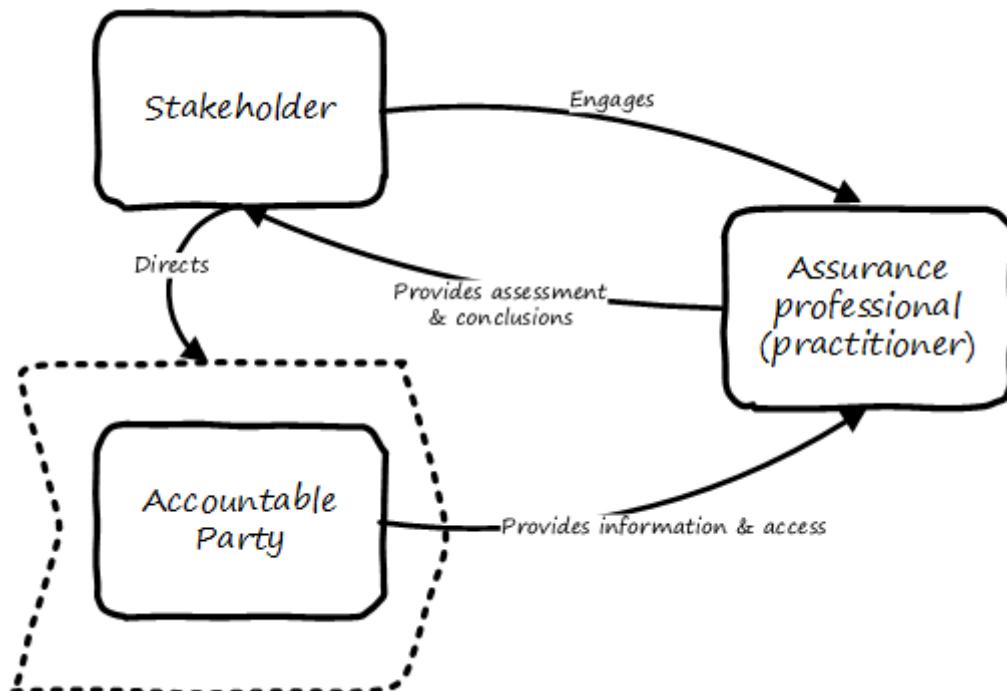
Assurance engagement—An engagement in which a practitioner aims to obtain sufficient appropriate evidence in order to express a conclusion designed to enhance the degree of confidence of the intended users other than the responsible party about the subject matter information (that is, the outcome of the measurement or evaluation of an underlying subject matter against criteria).

— ISAE 3000 (Revised), Assurance Engagements Other than Audits or Reviews of Historical Financial Information

Assurance means that, pursuant to an accountability relationship between two or more parties, an IT audit and assurance professional may be engaged to issue a written communication expressing a conclusion about the subject matters to the accountable party.

— Cobit 5 for Assurance

There are broader and narrower definitions of assurance. But all reflect some kind of three-party arrangement:



reflects concepts from [\[ISACA2013b\]](#) and [\[IAASB2013\]](#)

The above diagram is **one** common scenario:

The stakeholder (e.g. the audit committee of the board of directors) engages an assurance professional (e.g. an audit firm). The scope and approach of this is determined by the engaging party, although the accountable party in practice often has input as well.

The accountable party, at the direction responds to the assurance professional's inquiries on the audit topic.

The assurance professional provides the assessment back to the engaging party, and/or other users of the report (potentially including the accountable party).

This is a simplified view of what can be a more complex process and set of relationships. The ISAE3000 standard states that there must be at least three parties to any assurance engagement:

- The responsible (accountable) party
- The practitioner
- The intended users (of the assurance report)

But there may be additional parties:

- The engaging party
- The measuring/evaluating party (sometimes not the practitioner, who may be called on to render an opinion on someone **else's** measurement)

ISAE3000 goes on to stipulate a complex set of business rules for the allowable relationships between these parties [\[IAASB2013\]](#), pp 95-96. Perhaps the most important rule is that the practitioner **cannot** be the same as either the responsible party, or the intended users. There must be some level of professional objectivity.

What's the difference between assurance and simple consulting? There are two major factors:

- Consulting can be simply a two-party relationship - a manager hires someone for advice
- Consultants do not necessarily apply strong assessment criteria. Indeed, with complex problems, there may not be any such criteria. Assurance in general presupposes some existing standard of practice, or at least some benchmark external to the organization being assessed.

Finally, the concept of assurance criteria is key. Some assurance is executed against the responsible party's own criteria. In this form of assurance, the primary questions are "are you documenting what you do, and doing what you document?" That is, for example, do you have formal process management documentation (as discussed in [Chapter 9](#))? And are you following it?

Other forms of assurance use **external** criteria. A good example is the Uptime Institute's data center tier certification criteria, discussed below.

If criteria are weak or non-existent, the assurance engagement may be more correctly termed an advisory effort. Assurance requires clarity on this topic.

Types of assurance

Exercise caution in your business affairs; for the world is full of trickery.

— Max Ehrmann, Desiderata

The general topic of "assurance" implies a spectrum of activities.

In the strictest definitions, assurance is provided by licensed professionals under highly formalized arrangements. However, **while all audit is assurance, not all assurance is audit**. As noted in

COBIT for Assurance, "assurance also covers evaluation activities not governed by internal and/or external audit standards." [\[ISACA2013a\]](#) p15.

This is a blurry boundary in practice, as an assurance engagement may be undertaken by auditors, and then might be casually called an "audit" by the parties involved. And there is a spectrum of organizational activities that seem at least to be related to formal assurance:

- Brand assurance
- Quality assurance
- Vendor assurance
- Capability assessments
- Attestation services
- Certification services
- Compliance
- Risk management
- Benchmarking
- Other forms of "due diligence"

Some of these activities may be managed primarily internally, but even in the case of internally-managed activities, there is usually some sense of governance, some desire for objectivity.

From a purist perspective, internally directed assurance is a contradiction in terms. There is a conflict of interest in that in terms of the [three-party model](#) above, the accountable party is the practitioner.

However, it may well be less expensive for an organization to fund and sustain internal assurance capabilities and get much of the same benefits as from external parties. This requires sufficient organizational safeguards be instituted. Internal auditors typically report directly to the Board-level audit committee, and generally are not seen as having a conflict of interest.

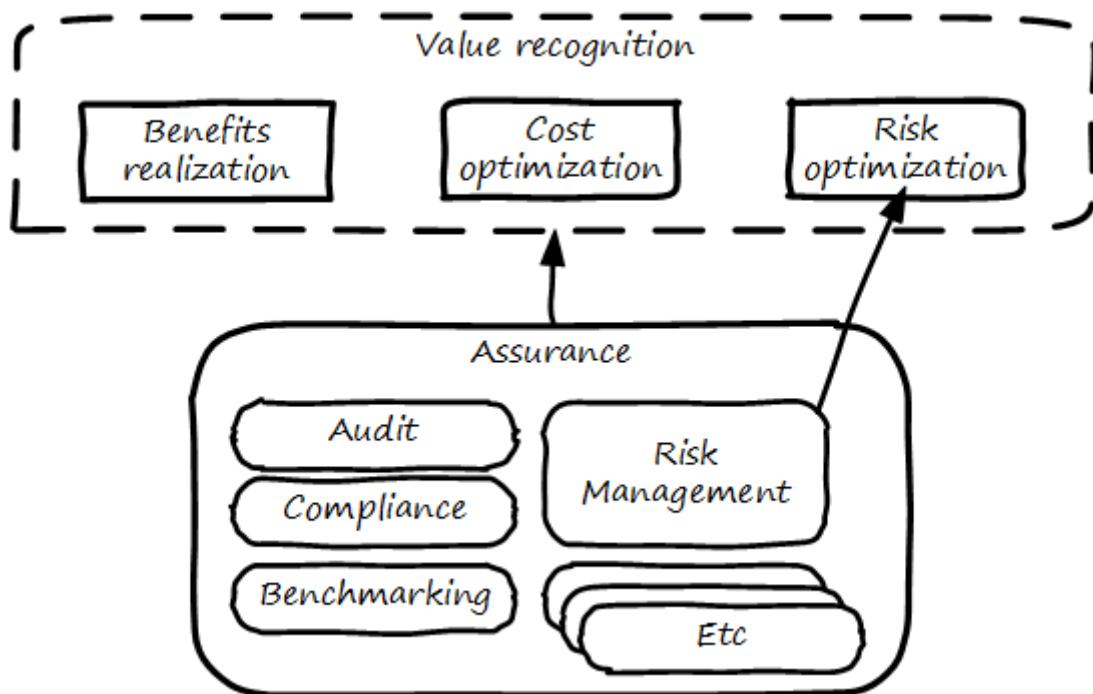
In another example, an internal compliance function might report to the corporate general counsel (chief lawyer), and not to any executive whose performance is judged based on their organization's compliance—this would be a conflict of interest. However, because the internal compliance function is ultimately under the CEO, their concerns can be overruled.

The various ways that internal and external assurance arrangements can work, and can go wrong, is a long history. If you are interested in the topic, review the histories of Enron, Worldcom, the 2008 mortgage crisis, and other such failures.

Assurance and risk management

Risk management (discussed in the [previous chapter section](#)) may be seen as part of a broader assurance ecosystem (for evidence of this, consider that the Institute of Internal Auditors offers a certificate in Risk Management Assurance). Assurance in practice may seem to be biased towards risk management, but (as with governance in general) assurance as a whole relates to all aspects of IT and digital governance, including effectiveness and efficiency.

Audit practices may be informed by known risks and particularly concerned with their mitigation, but risk management remains a distinct practice. Audits may have scope beyond risks, and audits are only one tool used by risk management.



In short, and as shown in the above diagram, assurance plays a role across value recognition, while risk management specifically targets the value recognition objective of risk optimization.

Non-audit assurance examples

Businesses must find a level of trust between each other . . . 3rd party reports provide that confidence. Those issuing the reports stake their name & liability with each issuance.

— James DeLuccia, Successfully Establishing and Representing DevOps in an Audit

Before we turn to a more detailed discussion of audit, we'll discuss some specifically non-audit examples of assurance seen in IT and digital management.

Example 1: Due diligence on a Cloud provider

Your company is considering a major move to Cloud infrastructure for its systems. The agility value proposition—the ability to minimize [Cost of Delay](#)—is compelling, and there may be some cost structure advantages as well.

But you are aware of some Cloud failures:

- In 2013, UK Cloud provider 2e2 went bankrupt and customers were given "24 to 48 hours to get ... data and systems out and into a new environment" [[duPreez2015](#)]. Subsequently, the provider demanded nearly £1 million pounds (roughly \$1.5 million) from its customers in order for their uninterrupted access to services (i.e., their data.) [[Venkatraman2013](#)]

- Also in 2013, Cloud storage provider Nirvanix went bankrupt and its customers also had a limited time to remove their data. MegaCloud went out of business with no warning two months later and all customers lost all data. [\[Butler2013\]](#), [\[Butler2014\]](#)
- In mid-2014, online source code repository Cloud Spaces (an early Github competitor) was taken over by hackers and destroyed. All data was lost. [\[Venezia2014\]](#), [\[Marks2014\]](#)

The question is, how do you manage the risks of trusting your data and organizational operations to a Cloud provider? This is not a new question, as computing has been outsourced to specialist firms for many years. You want to be sure that their operations meet certain standards:

- Financial standards
- Operational standards
- Security standards

Data center evaluations of cloud providers are a form of **assurance**. Two well known approaches are:

- The Uptime Institute's Tier Certification
- The American Institute of Certified Public Accountants' (AICPA) SOC 3 "Trust Services Report" certifying "Service Organizations" (based in turn on the SSAE-16 standard)

The Uptime Institute provides the well-known "Tier" concept for certifying data centers, from Tier I to Tier IV. In their words, "Certification provides assurances that there are not shortfalls or weak links anywhere in the data center infrastructure." [\[Uptime2016\]](#). The Tiers progress as follows [\[Uptime2014\]](#):

- Tier I: Basic Capacity
- Tier II: Redundant Capacity Components
- Tier III: Concurrently Maintainable
- Tier IV: Fault Tolerance

Uptime Institute certification is a generic form of assurance in terms of the [3-party model](#); the data center operator must work with the Uptime Institute who provides an independent opinion based on their criteria as to the data center's tier (and therefore effectiveness).

The SOC 3 report is considered an "assurance" standard as well. However, as mentioned above, this is the kind of "assurance" done in general by licensed auditors, and which might casually be called an "audit" by the participants. A qualified professional, again in the 3-party model, examines the data center in terms of the SSAE 16 reporting standard.

Your internal risk management organization might look to both Uptime Institute and SOC 3 certification as indicators that your Cloud provider risk is mitigated. (More on this in chapter section on Risk Management.)

Example 2: Internal process assessment

You may also have concerns about your internal operations. Perhaps your process for selecting technology vendors is unsatisfactory in general; it takes too long and yet vendors with critical

weaknesses have been selected. More generally, the actual practices of various areas in your organization may be assessed by external consultants using the related guidance:

- Enterprise Architecture with TOGAF
- Project Management with PMBOK
- IT processes such as Incident Management, Change Management, and Release Management with ITIL or CMMI-SVC

These assessments may be performed through using a maturity scale, e.g. CMM-derived. The CMM-influenced ISO/IEC 15504 standard may be used as a general process assessment framework. (Remember that we have discussed the [problems](#) with the fundamental CMM assumptions on which such assessments are based.)

According to [\[Bente2012\]](#), "In our own experience, we have seen that the maturity models have their limitations." They warn that maturity assessments of enterprise architecture at least are prone to being:

- Subjective,
- Academic,
- Easily manipulated,
- Bureaucratic,
- Superfluous, and
- Misleading.

Those issues may well apply to all forms of maturity assessments. Let the buyer beware. At least, the concept of maturity should be very carefully defined in a manner relevant to the organization being assessed.

Example 3: Competitive benchmarking

Finally, you may wonder, "how does my digital operation compare to other companies?" Now, it is difficult to go to a competitor and ask this. It's also not especially practical to go and find some non-competing company in a different industry you don't understand well. An entire industry has emerged to assist with this question.

We talked about the role of [industry analysts](#) in chapter 8. Benchmarking firms play a similar role, and in fact some analyst firms provide benchmarking services.

There are a variety of ways benchmarking is conducted, but it is similar to assurance in that it often follows the [3-party model](#). Some stakeholder directs an accountable party to be benchmarked within some defined scope. For example, the number of staff required to manage a given quantity of servers (aka admin:server) has been a popular benchmark. (Note that with cloud, virtualization, and containers, the usefulness of this metric is increasingly in question.)

An independent authority is retained. The benchmarker collects, or has collected, information on similar operations; for example, they may have collected data from 50 organizations of similar size on admin:server ratios. This data is aggregated and/or anonymized so that competitive concerns

are reduced. Wells Fargo will not be told "JP Morgan Chase has an overall ratio of 1:300;" they will be told "Average for financial services is 1:250."

In terms of formal assurance principles, the benchmark data becomes the assessment criteria. A single engagement might consider dozens of different metrics, and where simple quantitative ratios do not apply, the benchmarker may have a continuously maintained library of case studies for more qualitative analysis. This starts to shade into the kind of work also performed by industry analysts. As the work becomes more qualitative, it also becomes more advisory, and less about "assurance" per se.

Audit

The Committee therefore recommends that all listed companies should establish an audit committee.

— Cadbury Report

Agile or not, a team ultimately has to meet legal and essential organizational needs, and audits help to ensure this.

— Scott Ambler, Disciplined Agile Delivery

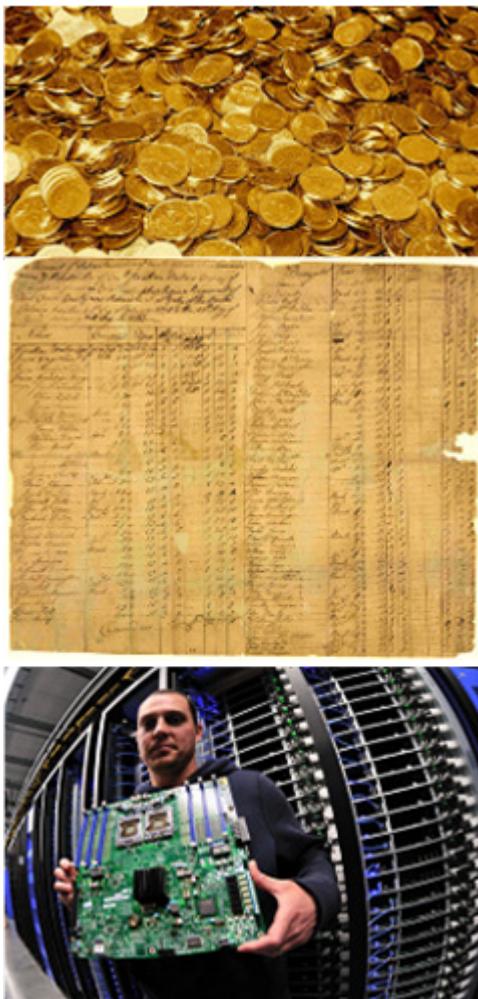
If you look up "audit" online or in a dictionary, you will see it mainly defined in terms of finance: an audit is a formal examination of an organization's finances (sometimes termed "books"). Auditors look for fraud and error, so that investors (like our [shop owner](#)) have confidence that accountable parties (e.g. the shop manager) are conducting business honestly and accurately.

Audit is critically important to the functioning of the modern economy, because there are great incentives for theft and fraud, and owners (in the form of shareholders) are remote from the business operations.

But what does all this have to do with information technology and digital transformation?

Digital organizations of course have budgets and must account for how they spend money. Since financial accounting and its associated audit practices are a well established practice, we won't discuss it here. (We discussed IT financial management and service accounting in [Chapter 8](#).)

Money represents a form of information, that of value. Money once was stored as precious metal. When carrying large amounts of precious metal became impossible, it was stored in banks and managed through paper record keeping.



Money, from physical to virtual

https://www.flickr.com/photos/tao_zhyn/442965594,

<https://www.flickr.com/photos/peagreenchick/396463634/>,

<https://www.flickr.com/photos/intelfreepress/6722296265/>, commercial use allowed for all

Paper record keeping migrated onto computing machines, which now represent the value once associated with gold and silver. Bank deposits (our [digital user's](#) bank account balance from Chapter 1) are now no more than a computer record—digital bits in memory—made meaningful by tradition and law, and secured through multiple layers of protection and assurance.

Because of this, auditors became increasingly interested in information technology. Clearly, these new electronic computers could be used to commit fraud in new and powerful ways. Auditors had to start asking, "How do you know the data in the computer is correct?"

This led to the formation in 1967 of the Electronic Data Processing Auditors Association (EDPAA), which eventually became ISACA (developer of [COBIT](#)).

It also became clear that computers and their associated operation were a notable source of cost and risk for the organization, even if they were not being directly used for financial accounting. This has led to the direct auditing of information technology practices and processes, as part of the broader assurance ecosystem we are discussing in this chapter section.

A wide variety of IT practices and processes may be audited. Auditors may take a general interest in whether the IT organization is "documenting what it does and doing what it documents" and

therefore this author has seen nearly every IT process audited.

IT auditors may audit projects, checking that the expected project methodology is being followed. They may audit IT performance reporting, such as claims of meeting Service Level Agreements. And they audit the organization's security approach - both its definition of security policies and controls, as well as their effectiveness.

External versus internal audit

There are two major kinds of auditors of interest to us:

- External auditors
- Internal auditors

Here is a definition of external auditor:

An external auditor is chartered by a regulatory authority to visit an enterprise or entity and to review and independently report the results of that review. [Moeller2013], p. 319.

Many accounting firms offer external audit services, and the largest accounting firms (such as PriceWaterhouse Coopers and Ernst & Young) provide audit services to the largest corporations. External auditors are usually certified public accountants, licensed by their state, and following industry standards (e.g. from the American Institute of Certified Public Accountants).

By contrast, internal auditing is housed internally to the organization, as defined by the Institute of Internal Auditors:

Internal auditing is an independent appraisal function established within an organization to examine and evaluate its activities as a service to the organization [Moeller2013], p. 320.

Internal audit is considered a distinct but complementary function to external audit. [Cadbury1992], 4.39. The internal audit function usually reports to audit committee. As with assurance in general, independence is critical - auditors must have organizational distance from those they are auditing, and must not be restricted in any way that could limit the effectiveness of their findings.

Audit practices

As with other forms of assurance, audit follows the [3-party model](#). There is a stakeholder, an accountable party, and an independent practitioner. The typical internal audit lifecycle consists of (derived from [ISACA2013a]):

- Planning/scoping
- Performing
- Communicating

In the scoping phase, the parties are identified (e.g. the board audit committee, the accountable and responsible parties, the auditors, and other stakeholders.)

The scope of the audit is very specifically established, including objectives, controls, and enablers

(e.g. processes) to be tested. Appropriate frameworks may be utilized as a basis for the audit, and/or the organization's own process documentation.

The audit is then performed.

A variety of techniques may be used by the auditors:

- Performance of processes or their steps
- Inspection of previous process cycles and their evidence (e.g. documents, recorded transactions, reports, logs, etc.)
- Interviews with staff
- Physical inspection or walkthroughs of facilities
- Direct inspection of system configurations and validation against expected guidelines
- Attempting what should be prevented (e.g. trying to access a secured system, or view data over the authorization level)

A fundamental principle is, "expected versus actual." There must be some expected result to a process step, a calculation, etc, that the actual result can be compared to.

Finally, the audit results are reported to the agreed users (often with a preliminary "heads up" cycle so that people are not surprised by the results). Deficiencies are identified in various ways, and typically are taken into system and process improvement projects.

Security

A measure of system's ability to resist unauthorized attempts at usage or behavior modification, while still providing service to legitimate users.

— Sandy Bacik, Enernex

In reality, organizations have many other things to do than practice security.

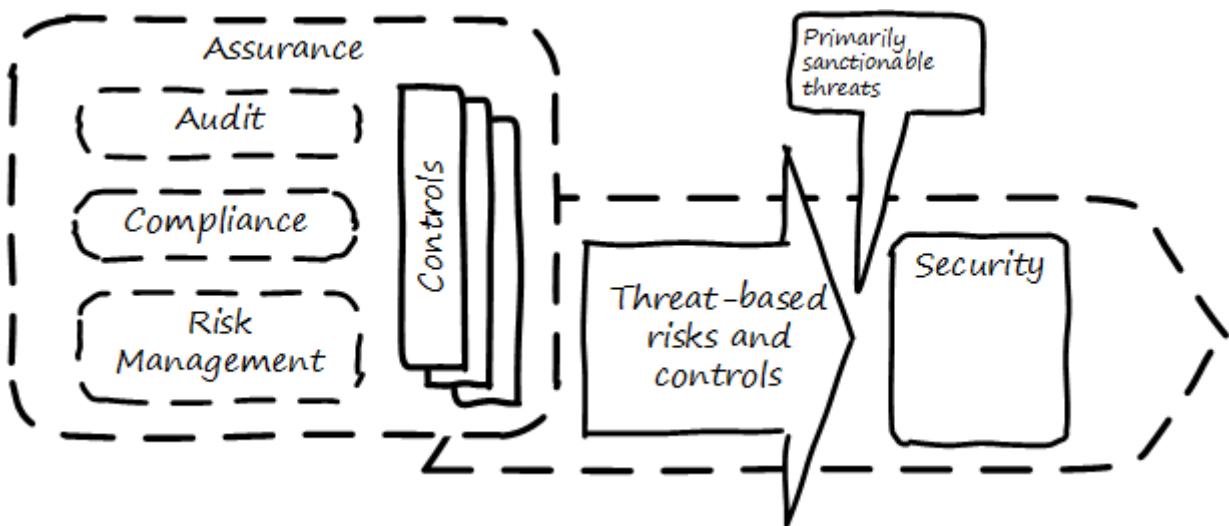
— Shon Harris, Guide to the CISSP

You have been practicing security since you first selected your initial choice of development platform in Chapter 2. But by now, your security capability is a well-established organization with processes spanning the enterprise, and a cross-functional steering committee composed of senior executives and with direct access to Board governance channels.

Security is a significant and well-known domain in and of itself. Ultimately, however, **security is an application of the governance and risk management principles discussed in the previous sections**. Deriving ultimately from the stakeholder's desire to sustain and protect the enterprise, security relies on

- accurate **risk** assessment,
- a clear **controls** strategy, and

- effective **assurance** practices (e.g. security **audits**).



So what distinguishes security from more general concepts of risk? The definition at the top of this chapter section is a good start, with its mention of "unauthorized attempts" to access or modify systems. The above graphic uses the term "sanctionable," meaning violations might lead to legal or at least organizational penalties.

Many risks might involve carelessness, or incompetence, or random technical failure, or accidents of nature. They might even involve fraud or misrepresentation. But security focuses on violations (primarily intentional, but also unintentional) of policies protecting organizational assets. In fact, "Assets Protection" is a common alternate name for corporate security.

"Authorization" is a key concept. Given some valuable resource, is access restricted to those who ought to have it? Are they who they say they are? Do they have the right to access what they claim is theirs? Are they conducting themselves in an expected and approved manner?

The security mentality is very different from the mentality found in a startup. A military analogy might be helpful. Being in a startup is like engaging in missions: search, extract, destroy, etc. One travels to a destination and operates with a single-point focus on completion.

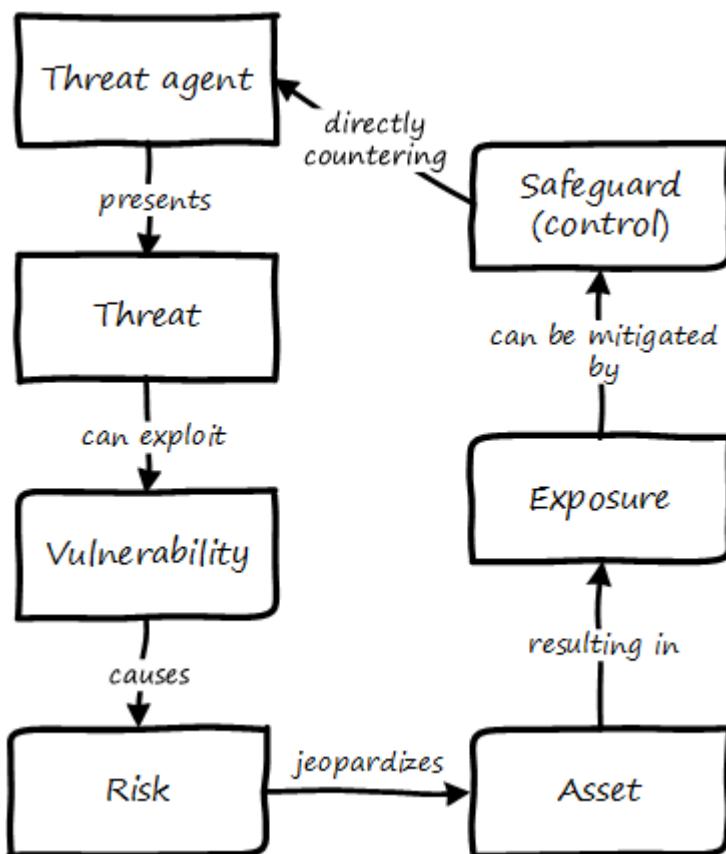
Security on the other hand is like defending a perimeter. You have to think broadly across a large area, assessing weaknesses and distributing limited resources where they will have the greatest effect.

This is where a systematic approach, including an accepted set of terminology, becomes key. The CISSP (Certified Information Systems Security Professional) guide proposes the following taxonomy:

- Vulnerability
- Threat agent
- Threat
- Risk
- Control

- Exposure
- Safeguard (e.g. control or enabler)

These terms are best understood in terms of their relationships, which are graphically illustrated in the diagram below:



(similar to [Harris2013], p. 27.)

In implementing controls, the primary principles are:

- Availability. The asset must be accessible to those entitled to it.
- Integrity. The asset must be protected from destruction or corruption.
- Confidentiality. The asset must not be accessible to those not entitled to it.

Information classification

At a time when the significance of information and related technologies is increasing in every aspect of business and public life, the need to mitigate information risk, which includes protecting information and related IT assets from ever-changing threats, is constantly intensifying.

— ISACA, Cobit 5 for Security

Before we turn to security engineering and security operations, we need to understand the business

context of security. The assets at risk are an important factor, and risk management gives us a good general framework. One additional technique associated with security is information classification. A basic hierarchy is often used, such as:

- Public
- Internal
- Confidential
- Restricted

The military uses the well known levels of:

- Unclassified
- Confidential
- Secret
- Top Secret

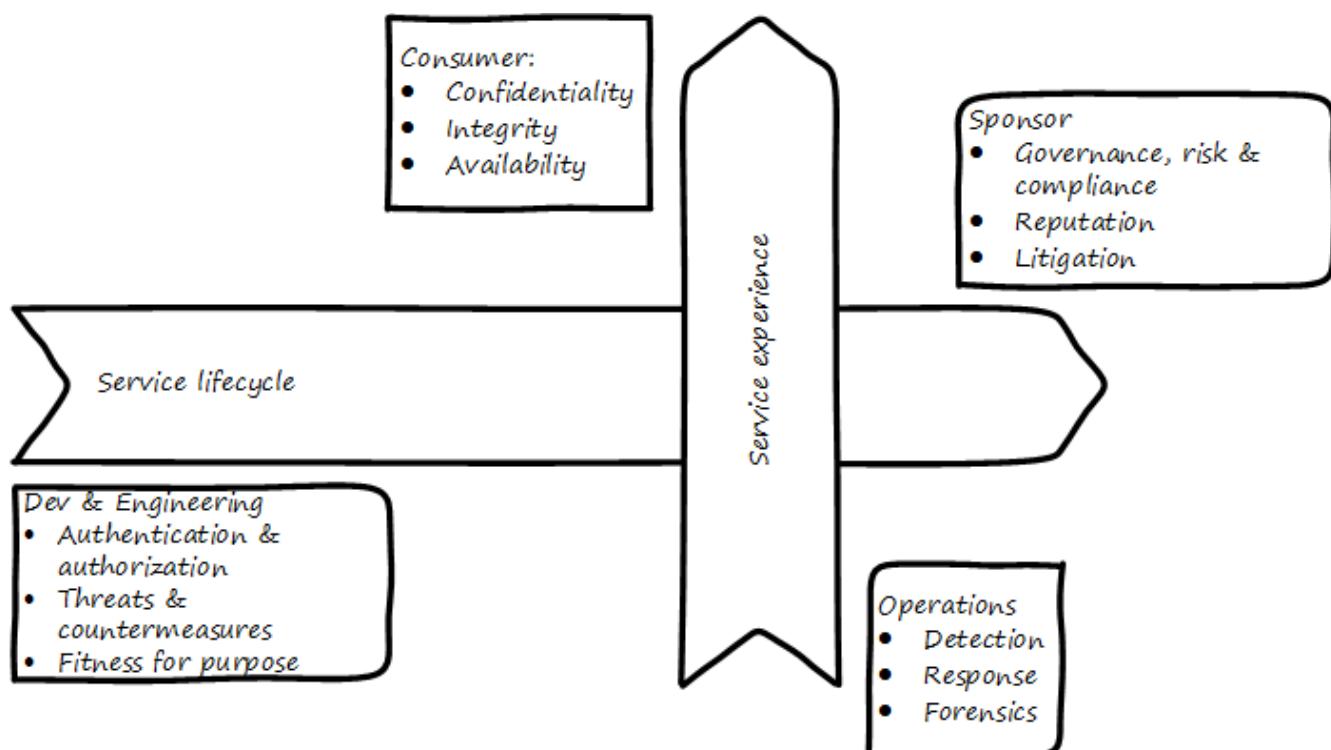
These classifications assist in establishing the security risk and necessary controls for a given digital system and/or process.

Information also can be categorized by subject area. This becomes important from a compliance point of view. This will be discussed in Chapter 11, in the chapter section on Records Management.

Security engineering

For the next two sections, we will adopt a "2-axis" view, first proposed in [Betz2011a].

In this model, the systems lifecycle is considered along the horizontal access, and the user experience is considered along the vertical access (which also maps to the "stack.") In the following picture, we see the distinct concerns of the various stakeholders in the dual-axis model:



Consumer versus sponsor perspective

The consumer of the digital service has different concerns from the sponsor/customer (in our [3-party model](#)). The consumer (our woman [checking her bank balance](#)) is concerned with immediate aspects of confidentiality, integrity, and availability:

- Is this communication private?
- Is my money secure?
- Can I view my balance and do other operations with it (e.g. transfer it) confident of no interference?

The sponsor on the other hand has derivative concerns:

- Are we safe from the bad publicity that would result from a breach?
- Are compliant with laws and regulations, or are we risking penalties for non-compliance (as well as risking security issues?)
- Are our security activities as cost-efficient as possible, given our risk appetite?

Security architecture

Security engineering is concerned with the fundamental security capabilities of the system, as well as ensuring that any initial principles established for the system are adhered to as development proceeds, and/or as vendors are selected and perhaps replaced over time.

There are multitudes of books written on security from an engineering, architecture and development perspective. The tools, techniques, and capabilities evolve quickly every year, which is why having a fundamental business understanding based in a stable framework of risk and control is essential.

This is a book on management, so we are not covering technical security practices and principles, any more than we are covering specific programming languages or distributed systems engineering specifics. Studying for the Certified Information Systems Security Professional exam will provide both an understanding of security management, as well as current technical topics. A glance at the CISSP guide shows how involved such topics can be:

- The Harrison-Rizzo-Ullman security model
- The Diffie-Hellman Asymmetrical Encryption Algorithm
- Functions and Protocols in the OSI Model

Again, the issue is mapping such technical topics to the fundamentals of risk and control. Key topics we note here include:

- Authentication & authorization
- Network security
- Cryptography

Authentication and authorization are the cornerstones of **access**, i.e. the gateway to the asset. **Authentication** confirms that a person is who they say they are. **Authorization** is the management

of their access rights (can they see the payroll? reset others' passwords?)

Network security is a complex sub-domain in and of itself. Because attacks typically transpire over the Internet and/or internal organizational networks, the structure and capabilities of networks are of critical concern, including topics such as:

- Routing
- Firewalls
- the Domain Name Service

Finally, **cryptography** is the "storage and transmission of data in a form that only those it is intended for can read and process" [\[Harris2013\]](#).

All of these topics, and many more, require in depth study and staff development. At this writing (mid-2016), there is a notable shortage of skilled security professionals. Therefore, a critical risk is that your organization might not be able to hire people with the needed skills (consider our section on [resource management](#))

Security and SDLC

Security is a concern throughout the application lifecycle. You already know this, otherwise you would not have reached enterprise scale. But now you need to formalize it with some consistency, as that is what regulators and auditors expect, and it also makes it easier for your staff to work on various systems.

Security should be considered throughout the SDLC, including systems design, but this is easier said than done. Organizations will always be more interested in a system's functionality than its security. However, a security breach can ruin a company.

The CISSP recommends (among other topics) consideration of the following throughout the systems lifecycle:

- The role of environmental (e.g. operating system-level) safeguards versus internal application controls
- The challenges of testing security functionality
- Default implementation issues
- Ongoing monitoring

Increasingly important controls during the construction process in particular are:

- Code reviews
- Automated code analysis

We discuss the Netflix [Simian Army](#)—an important automated tool that can serve as a control—in the next chapter section.

Sourcing and security

Vendors come and go in the digital marketplace, offering thousands of software-based products across every domain of interest (we call this the [technology product lifecycle](#)). Inevitably, these products have security errors. A vendor may issue a "patch" for such an error, which must be applied to all instances of the running software. Such patches are not without risk, and may break existing systems; they therefore require testing under conditions of urgency.

Increasingly, software is offered as a service, in which case it is the vendor responsibility to patch their own code. But what if they are slow to do this? Any customer relying on their service is running risk, and other controls may be required to mitigate the exposure.

One important source of vulnerabilities is the [National Vulnerability Database](#) supported by the National Institute for Standards and Technology. In this database, you can look up various products and see if they have known security holes. Using NVD is complex and not something that can be simply and easily "implemented" in a given environment, but it does represent an important, free, taxpayer-supported resource of use to security managers.

An important type of vulnerability is the "zero-day" vulnerability. With this kind of vulnerability, knowledge of a security "hole" becomes widespread before any patches are available (i.e., the software's author and users have "zero days" to create and deploy a fix). Zero-day exploits require fast and aggressive application of alternate controls, which leads us to the topic of security operations.

Security operations

Networks and computing environments are evolving entities; just because they are secure one week does not mean they are secure three weeks later.

— Shon Harris, Guide to the CISSP

Security requires ongoing [operational attention](#). Security operations is first and foremost a form of operations, as discussed in [Chapter 6](#). It requires on-duty and on-call personnel, and some physical or virtual point of shared awareness (for example, a physical Security Operations Center, perhaps co-located with a Network Operations Center). Beyond the visible presence of a Security Operations Center, various activities must be sustained. These can be categorized into four major areas:

- Prevention
- Detection
- Response
- Forensics

Prevention

An organization's understanding for what constitutes a "secure" system is continually evolving.

New threats continually emerge and the alert security administrator has an ongoing firehose of bulletins, alerts, patch notifications, and the like to keep abreast of.

These inputs must be synthesized by an organization's security team into a set of security standards for what constitutes a satisfactorily-configured ("hardened") system. Ideally, such standards are automated into policy-driven systems configuration approaches; in less ideal situations, manual configuration—and double-checking—is required.

Prevention activities include:

- maintaining signatures for intrusion detection and anti-virus systems
- software patching (e.g. driven by the [technology product lifecycle](#) and updates to the [National Vulnerability Database](#))
- ongoing maintenance of user authorizations and authentication levels
- ongoing testing of security controls (e.g., firewalls, configurations, etc.)
- updating security controls appropriately for new or changed systems

Detection

There are many kinds of events that might indicate some security issue; systems exposed to the open Internet are continually scanned by a wide variety of often-hostile actors. Internal events, such as unscheduled/unexplained system restarts, may also indicate security issues. The challenge with security monitoring is identifying patterns indicating more advanced or persistent threats. When formalized, such patterns are called "signatures."

One particular form of event that can be identified for systems under management are configuration state changes.

For example, if a core operating system file—one that is well known and not expected to change—changes in size one day with no explanation, this might be indicative of a security exploit. Perhaps an attacker has substituted this file with one containing a "backdoor" allowing access. Tools such as Tripwire are deployed to scan and inventory such files and key information about them ("metadata") and raise alerts if unexpected changes occur. Infrastructure managers such as Chef and Puppet may also serve as inputs into security event management systems; for example, they may detect attempts to alter critical configuration files and in their re-converging the managed resource back to its desired state, can be a source of valuable information about potential exploits. Such tools also may be cited as controls for various kinds of security risks.

We have discussed the importance of configuration management in both Chapter 2 and Chapter 6. In Chapter 2, we discussed the important concept of [Infrastructure as Code](#) and [policy-driven configuration management](#); we revisited the importance of configuration management from an operational perspective in [Chapter 6](#). Configuration management also will re-appear in Chapters 11 and 12.

IMPORTANT

It should be clear by now that configuration management is one of the most critical enabling capabilities for digital management, regardless of whether you look to traditional IT service management practices or modern DevOps approaches.

Detection activities include:

- monitoring events and alerts from intrusion detection and related operational systems
- Analyzing logs and other artifacts for evidence of exploits

Response

Security incidents require responses. Activities include:

- Declaring security incidents
- Marshalling resources (staff, consultants, law enforcement) to combat
- Developing immediate tactical understanding of the situation
- Developing a response plan, under time constraints
- Executing the plan, including ongoing monitoring of effectiveness and tactical correction as needed
- Keeping stakeholders informed as to situation

Forensics

Finally, security incidents require careful after-the-fact analysis:

- Analyzing logs and other artifacts for evidence of exploits
- Researching security incidents to establish causal factors and develop new preventative approaches (thus closing the loop)

Relationship to other processes

As with operations as a whole, there is ongoing monitoring and reporting to various stakeholders, and interaction with other processes.

One of the most important operational processes from a security perspective is Change Management. Configuration state changes (potentially indicating an exploit in progress) should be reconciled first to Change Management records. Security response may also require emergency Change processes. ITSM Event and Incident Management may be leveraged as well.

NOTE

The particular concerns of security may interfere with cross-process coordination.
This is a topic beyond the scope of this book.

Security and assurance

Quis custodiet ipsos custodes?

— Latin for "Who watches the watchers?"

Given the critical importance of security in digital organizations, it is an essential matter for governance attention at the highest levels.

Security management professionals are accountable to governance concerns just as any other

manager in the digital organization. Security policies, processes, and standards are frequently audited, by both internal auditors as well as external [assurance](#) professionals (not only auditors, but other forms of assurance as well).

The idea that an "Assets Protection" group might itself be audited may be hard to understand, but security organizations such as police organizations have Internal Affairs units for just such purposes.

Security auditors might review the [security processes](#) mentioned above, or system configuration baselines, or log files, or any number of other artifacts, depending on the goals and scope of a security audit. Actual penetration testing is a frequently used approach: the hiring of skilled "white-hat" hackers to probe an organization's defenses. Such hackers might be given license to probe as far as they possibly can and return with comprehensive evidence of what they were able to access (customer records, payrolls, account numbers and balances, etc.)

acronym: DFIR, Digital Forensics and Incident Response

Digital Governance

...transforming to an agile delivery approach is difficult. Companies or government agencies that have been in existence for decades have built up layers of deep and wide waste, many of them around governance.

Business or IT governance constructs and processes that do not support new technology-centric go-to-market models and products must be either replaced or substantially evolved; otherwise, they **increase** risk on the agile projects [emphasis added].

— Gorans/Ambler, Disciplined Agile Delivery

As with other chapters, we have presented the material defining this chapter's topic "on its own terms." Certainly, there is little chance that the core principles of governance will cease to have importance, even in the new digital economy.

However, as Ambler and Gorans note above, the legacy of IT governance is wide, deep, and often wasteful. Approaches based on mis-applied [Taylorism](#) and misguided, CMM-inspired [statistical process control](#) have resulted in the creation of ineffective, large-scale [IT bureaucracies](#) whose sole mission seems to be the creation and exchange of non-value-add [secondary artifacts](#), while lacking any clear concept of an [execution model](#).

What is to be done? Governance will not disappear any time soon. Simply arguing against governance (ala [NoEstimates](#)) is unlikely to succeed.

Instead, this book argues the most effective answer lies in a re-examination of the true concerns of governance:

- sustaining innovation and effective value delivery

- maintaining efficiency
- optimizing risk

These fundamental principles ("top-line," "bottom-line," "risk") define value for organizations around the world, whether for-profit, non-profit, or governmental. After considering the failings of IT governance, we'll re-examine it in light of these objectives and come up with some new answers for the digital era.

The failings of IT governance

...many GRC management processes within enterprises are designed and implemented within a command-and-control paradigm. They are highly centralized and are viewed as the purview of specialized GRC teams, who are not held accountable for the outcomes of the processes they mandate. The processes and controls these teams decree are often derived from popular frameworks without regard to the context in which they will be applied and without considering their impact on the entire value stream of the work they affect. They often fail to keep pace with technology changes and capabilities that would allow the desired outcomes to be achieved by more lightweight and responsive means. This forces delivery teams to complete activities adding no overall value, create bottlenecks, and increase the overall risk of failure to deliver in a timely manner.

— Jez Humble et al, Lean Enterprise

From the perspective of digital transformation, there are many issues with traditional IT governance and the assumptions and practices characterizing it.

The fundamental lesson of digital transformation

Consider the idea of "[programmability](#)" mentioned at the start of this chapter. A highly "programmable" position is one where the responsibilities can be specified in terms of their activities. **And what is the fundamental reality of digital transformation?** It is no accident that such positions are called "programmable." In fact, they **are** being "programmed away" or "eaten by software"- leaving only higher-skill positions that are best managed by objective, and which are more sensitive to cultural dynamics.

Preoccupation with "efficiency" fades as a result of the decreasingly "programmable" component of work. The term "efficiency" signals a process that has been well defined (is "programmable") to the point where it is repeatable and scalable. Such processes are ripe for automation, commoditization, and outsourcing, and this is in fact happening.

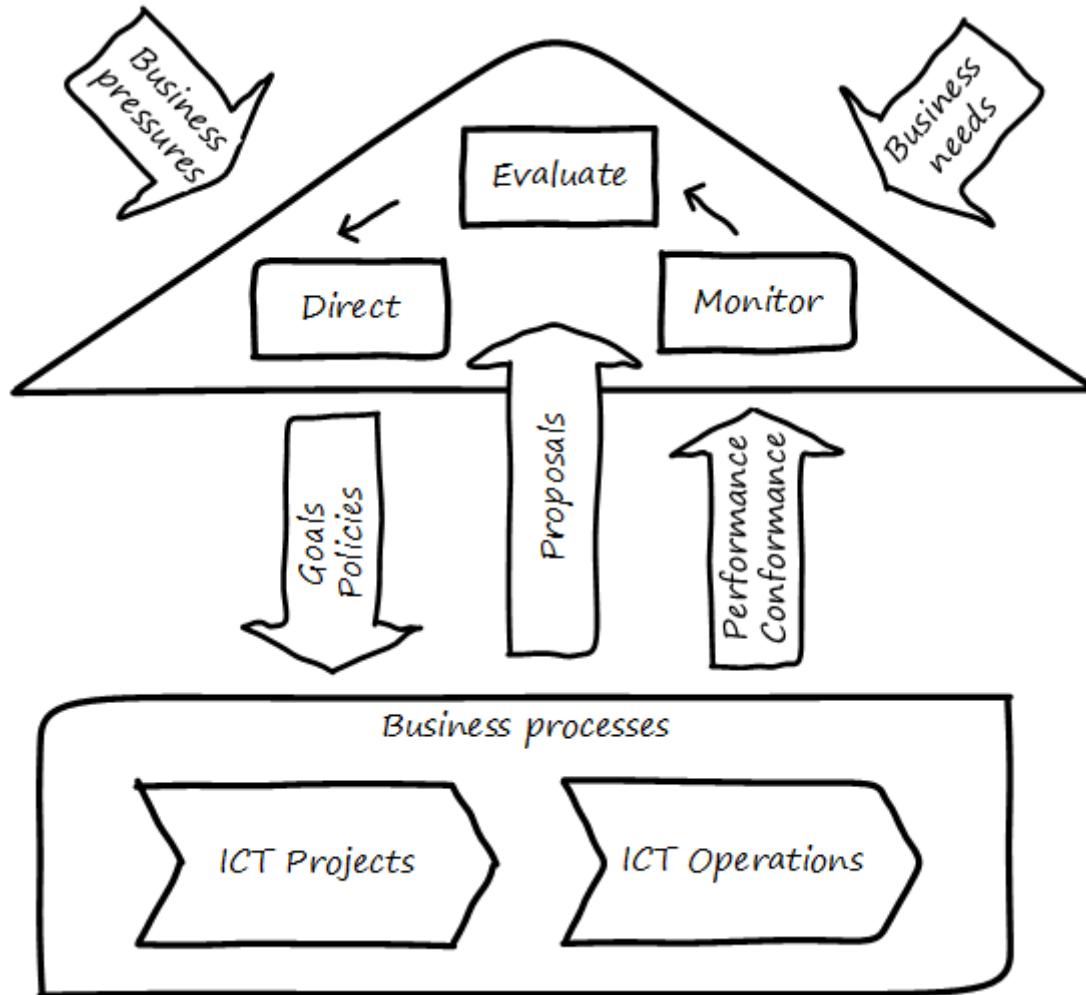
If the repetitive process is retained within an organization, the drive for efficiency leads to automation, and eventually efficiency is expressed through concerns for capacity management and the consumption of computing resources.

And when such repetitive concerns are **not** retained by the organization, but instead become a

matter of sourcing rather than execution, the emphasis shifts to risk management and governance of the supplier.

The remaining uncertain and creative processes **should not just be managed for "efficiency"** and need to be managed for effectiveness, including fast feedback, collaboration, culture, and so forth.

Project versus operations as operating model



similar to [\[ISO2008\]](#)

As we can see in the above diagram, ISO/IEC 38500 assumes a specific IT operating model, one in which projects are distinct from operations. We have discussed the growing influence of product-centric digital management throughout this book, but as of 2016 the major IT governance standard still does not recognize it.

The ripple effects are seen throughout other guidance and commentary. In particular, the project-centric mindset is closely aligned to the idea of IT and the CIO as primarily order-takers.

CIO as order-taker

Throughout much of the IT governance guidance, certain assumptions become evident:

- There is an entity that can be called "The Business."
- There is a distinct entity called "IT" (for "Information Technology").

- It is the job of "IT" to take direction (i.e. orders) from "The Business" and to fulfill them.
- There is a significant risk that "IT" activities (and by extension, dollars spent on them) may **not** be correctly allocated to the preferred priorities of "The Business." IT may spend money unwisely, on technology for its own sake. This risk needs to be controlled.
- The needs of "The Business" can be precisely defined and it is possible for "IT" to support those needs with a high degree of predictability as to time and cost. This predictability is assumed even when those activities imply multi-million dollar investments and months or years of complex implementation activities.
- When such activities do not proceed according to initial assumptions, this is labeled an "IT failure." It is assumed that the failure could have been prevented through more competent management, "rigorous" process, or diligent governance, especially on the IT side.

There may be organizations where these are reasonable assumptions. (This book does not claim they do not exist.) But there is substantial evidence for the existence of organizations for whom these assumptions are untrue.

The fallacies of "rigor" and repeatability

...it takes more time than you have to prove less than you'd like.

— Cem Kaner et al, Testing Computer Software

Rigor? Or rigor mortis?

— anonymous

One of the most critical, yet poorly understood facts of software development and by extension complex digital system innovation is the impossibility of "rigor."

Software engineers are taught early that "completely" testing software is impossible [Kaner1999]; yet it seems that this simple fact (grounded in fundamentals of computer science and information theory) is not understood by many managers.

A corollary fallacy is that of repeatable processes, when complexity is involved. We may be able to understand repeatability at a higher level, through approaches like [case management](#) or the Checklist Manifesto's [submittal schedules](#), but process control in a formal sense is simply [impossible](#), and the quest for it is essentially [cargo cult](#) management.

And yet quotes like the following are common in IT governance discussions:

...the questions a senior manager should ask include: "How good are my IT governance processes at effectively delivering strategic business value year after year?" and "Are my processes repeatable, predictable, and scalable, and are they truly meeting the needs of my business (outside of IT) and my customers?" [Moeller2008], p. 6.

With all due respect to the author, value that can be delivered "repeatably," "year after year" is for the most part commodity production, not innovative [product development](#). Strategy is notably

difficult to commoditize...

Another way to view this is in terms of the [decline of traditional IT](#). As you review those diagrams, understand that much of IT governance has emerged from the arguably futile effort to deliver product innovation in a low-risk, "efficient" manner. This desire has led, as Ambler and Gorans note at the top of this chapter section, to the creation of layers and layers of bureaucracy and [secondary artifacts](#).

The cynical term for this is "theater," as in an act that is essential unreal, but presented for the entertainment and distraction of an audience.

As we noted above, a central reality of digital transformation is that commoditized, predictable, programmable, repeatable, "efficient" activities are being quickly automated, leaving governance to focus more on effectiveness of innovation (e.g. product development) and management of supplier risk. Elaborate IT operating models specifying hundreds of interactions and deliverables, in a futile quest for "rigor" and "predictability," are increasingly relics of another time.

Digital effectiveness

Let's return to the first value objective: effectiveness.

We define effectiveness as "top-line" benefits: new revenues and preserved revenues. New revenues may come from product innovation, as well as successful marketing and sales of existing products to new markets (which itself is a form of innovation).

Traditionally, "back-office" information technology was rarely seen as something contributing to effectiveness, innovation, and top-line revenue. Instead, the first computers were used to increase [efficiency](#), through automating clerical work. The same processes and objectives could be executed for less money, but they were still the same back-office processes.

With digital transformation, product innovation and effectiveness is now a much more important driver. Yet product-centric management is still poorly addressed by traditional IT governance, with its emphasis on distinguishing projects from operations.

One tool that becomes increasingly important is a portfolio view. While project management offices may use a concept of "portfolio" to describe temporary initiatives, such project portfolios rarely extend to tracking ongoing operational benefits. Alternative approaches also should be considered such as the idea of an [options approach](#).

Digital efficiency

Efficiency is a specific, technical term, and although often inappropriately prioritized, is always an important concern. Even a digitally-transforming, product-centric organization can still have governance objectives of optimizing efficiency. Here are some thoughts on how to re-interpret the concept of efficiency.

Consolidate the pipelines

One way in which digital organizations can become more efficient is to consolidate development as much as possible into common pipelines. Traditionally, application teams have owned their own

development and deployment pipelines, at the cost of great, non-value add variability. Even centralizing source control has been difficult.

This is challenging for organizations with large legacy environments, but full-lifecycle pipeline automation is becoming well understood across various environments (including the mainframe).

Reduce internal service friction

Another way of increasing efficiency is to standardize integration protocols across internal services, ala [Amazon](#). This reduces the need for detailed analysis of system interaction approaches every time two systems need to exchange data. This is a form of reducing transaction costs and therefore consistent with Coase's theory of the firm [[Coase1937](#)]..

Within the boundary of a firm, collaboration between internal services should be easier because of reduced transaction costs. It's not hard to see that this would be the case for digital organizations: security, accounting, customer relationship management would all be more challenging and expensive for externally-facing services.

However, since a firm is a system, a service within the boundaries of a firm will have more constraints than a service constrained only by the market. The internal service may be essential to other, larger-scoped services, and may derive its identity primarily from that context.

Because the need for the service is well-understood, the engineering risk associated with the service may also be reduced. It may be more of a component than a product. See the parable of the [the Flower and the Cog](#). Reducing service redundancy is a key efficiency goal within the bounds of a system — more to come on this in Chapter 12.

Manage the process portfolio

Processes require ongoing scrutiny. The term "organizational scar tissue" is used when specific situations result in new processes and policies, that in turn increase transactional friction and reduce efficiency throughout the organization.

Processes can be consolidated, especially if specific procedural detail can be removed in favor of larger-grained [case management](#) or [Checklist Manifesto](#) concepts including the [submittal schedule](#). As part of eventual automation and digital transformation, processes can be ranked as to how "heavyweight" they are. A typical hierarchy, from "heaviest" to "lightest," might be:

- Project
- Release
- Change
- Service request
- Automated self-service

The organization might ask itself:

- Do we need to manage this as a project? Why not just a release?
- Do we need to manage this as a release? Why not just a change?

- Do we need to manage this as a change? Why not just a service request?
- Do we need to manage this as a service request? Why is it not fully automated self-service?

As we saw in our examination of the [Chubby locking service](#), there may be good reason to retain some formality. The point is to keep asking the question. Do we **really** need a separate process? Or can the objectives be achieved as part of an existing process or other enabler?

Governance as demand

A steam engine's governor imposes some load, some resistance, on the engine. In the same way, governance activities and objectives, unless fully executed by the directing body (e.g. the board), themselves impose demand on the organization.

This points to the importance of having a clear [demand/execution framework](#) in place to manage governance demand. The organization does not have an unlimited capacity for audit response, reporting, and the like. In order to understand the organization as a system, governance demand needs to be tracked and accounted for, and challenged for efficiency just as any other sort of demand.

Leveraging the digital pipeline

Finally, efficiency asks: can we leverage the digital pipeline itself to achieve governance objectives? This is not a new idea. The governance/management interface must be realized via specific enablers, such as processes. Processes can (and often should) be automated. Automation is the *raison d'être* of the digital pipeline; if the process can be expressed as user stories, behavior-driven design, or other forms of requirement, it simply is one more state change moving from dev to ops.

In some cases, the governance stories must be applied to the pipeline itself. This is perhaps more challenging, but there is no reason the pipeline itself cannot be represented as code and managed using the same techniques.

The automated enablers then can report their status up to the Monitoring activity of governance, closing the loop. Auditors should periodically re-assess their effectiveness.

Digital risk management

Poorly governed and managed information and technology will destroy value or fail to deliver benefits...

— COBIT 5 for Risk

Finally, from an IT governance perspective, what is the role of IT risk management in the new digital world? It's not that risk management goes away. Many risks that are well understood today, will remain risks for the foreseeable future. But there are significant new classes of risk that need to be better understood and managed:

- Unmanaged demand and disorganized execution models leading to multi-tasking, which is destructive of value and results

- High queue wait states, resulting in uncompetitive speed to deliver value
- Slow feedback due to large batch sizes, reducing effectiveness of product discovery
- New forms of supplier risk, as services become complex composites spanning the Internet ecosystem.
- Toxic cultural dynamics destructive of high team performance
- Failure to incorporate cost of delay in resource allocation and work prioritization decisions

All of these conditions can reduce or destroy revenues, erode organizational effectiveness, and worse. It is hard to see them as other than risks, yet there is little attention to such factors in the current (as of late 2016) "best practice" guidance on risk.

Cost of delay as risk

In today's digital governance there must be a greater concern for outcome and effectiveness, especially in terms of time to market (minimizing [Cost of Delay](#)). Previously, concerns for efficiency might lead a company to overburden its staff, resulting in queuing gridlock, too much work in process, destructive multitasking, and ultimately failure to deliver timely results (or deliver at all).

Such failure to deliver was tolerated because it seemed to be a common problem across most IT departments, and because digital transformation had not taken hold yet. IT systems were often back office and delays in delivering them (or significant issues in their operation) were not **quite** as damaging.

Now, effectiveness of delivery is essential. The interesting, and to some degree unexpected result, is that both efficiency and risk seem to be benefiting as well. Cross-functional, focused teams are both more effective and more efficient, and able to manage risk better as well. Systems are being built with both increased rapidity as well as improved stability, and the automation enabling this provides robust audit support.

Team dynamics as risk

We've covered culture in some depth in Chapter 7. Briefly, from a governance perspective:

The importance of organizational culture has been discussed by management thinkers since at least W.E. Deming. In a quote often attributed to Peter Drucker, "culture eats strategy for breakfast." But it has been difficult at best to quantify what we mean by culture.

Quantify? Some might even say quantification is impossible. But Google and the State of DevOps research have done so. Google has established the importance of psychological safety in forming effective, high-performing teams [\[Rozovsky2015\]](#). And the State of DevOps research, applying the Westrum typology, has similarly confirmed that pathological, controlling cultures are destructive of digital value [\[Puppet2015\]](#).

These facts should be taken seriously in digital governance discussions. So-called "toxic" leadership (an increasing concern in the military itself [\[Vergun2015\]](#)) is destructive of organizational goals and stakeholder value. It can be measured and managed, and should be a matter of attention at the highest levels of organizational governance.

Sourcing and SIAM risk

We have already covered contracting in terms of software and Cloud. But in terms of the emergence model, it is typical that companies enter into contracts before having a fully mature sourcing and contract management capability with input from the governance, risk, and compliance perspective.

We've touched on the issues of [Cloud due diligence](#) and [sourcing and security](#) in this chapter, and [supplier integration and management](#) in chapter 8. The 2e2 case discussed is interesting; it seems that due diligence had actually been performed. Additional controls could have made a key difference, in particular [business continuity planning](#).

There are a wide variety of supplier-side risks that must be managed in Cloud contracts:

- Access
- Compliance
- Data location
- Multi-tenancy
- Recovery
- Investigation
- Viability (assurance)
- Escrow

We've emphasized throughout this book the dynamic nature of digital services. This presents a challenge for risk management of digital suppliers. This year's audit is only a point-in-time snapshot; how to maintain assurance with a fast-evolving supplier? This leading edge of Cloud sourcing is represented in discussions such as "Dynamic certification of Cloud services: Trust, but verify!":

the on-demand, automated, location-independent, elastic, and multi-tenant nature of cloud computing systems is in contradiction with the static, manual, and human process-oriented evaluation and certification process designed for traditional IT systems...

Common certificates are a backward look at the fulfillment of technical and organizational measures at the time of issue and therefore represent a snapshot. This creates a gap between the common certification of one to three years and the high dynamics of the market for cloud services and providers.

The proposed dynamic certification approach adopts the common certification process to the increased flexibility and dynamics of cloud computing environments through using of automation potential of security controls and continuous proof of the certification status [Lins2016].

It seems likely that such ongoing dynamic evaluation of cloud suppliers would require something akin to [Simian Army](#) techniques, discussed below.

Beyond increasing supply-side dynamism, risk management in a full SIAM (Supplier Integration and Management) sense is compounded by the complex interdependencies of the services involved.

All of the Cloud contracting risks need to be covered, as well as further questions such as:

- If a given service depends on two sub-services ("underpinning contracts"), what are the risks for the failure of either or both of the underpinning services? What are the controls?

Automating digital governance

Digital exhaust

One governance principle we will suggest here is to develop a governance architecture as an inherent part of the delivery system, not as an additional set of activities. We use the concept of "digital exhaust" to reinforce this.

What is "digital exhaust"?

Digital exhaust, for the purposes of this book, consists of the extraneous data, and information that can be gleaned from it, originating from the development and delivery of IT services.

Consider an automobile's exhaust. It does not help you get to where you are going, but it's an inevitable aspect of having an internal combustion engine. Since you have it, you can monitor it and gain certain insights as to whether your engine is running efficiently and effectively. You might even be able to identify if you are at risk of an engine failure.

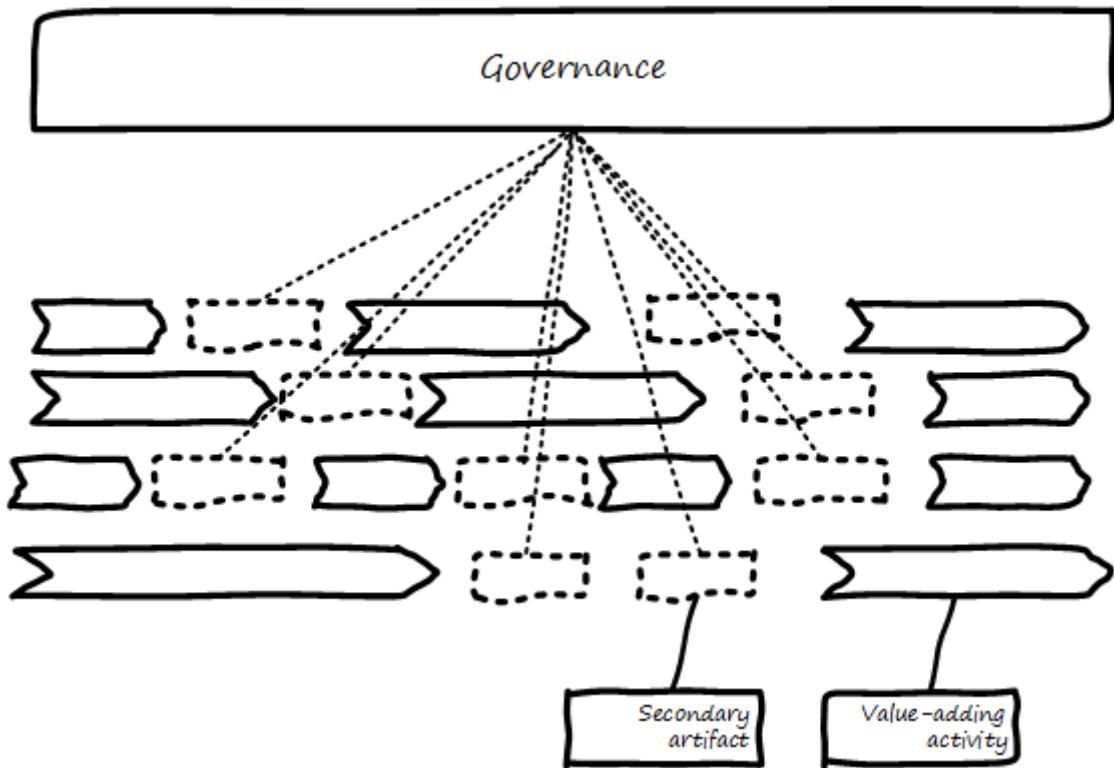
The term "digital exhaust" is also applied to the data generated from the Internet of Things. This usage is conceptually aligned to our usage here, but somewhat different in emphasis.

To leverage digital exhaust, focus on the critical, always-present systems that enable digital delivery:

- In chapter 2, we introduced the concept of [version control](#)
- In chapter 3 we introduced the idea of a [continuous delivery pipeline](#)
- In chapter 6 we introduced [monitoring](#) as part of operations

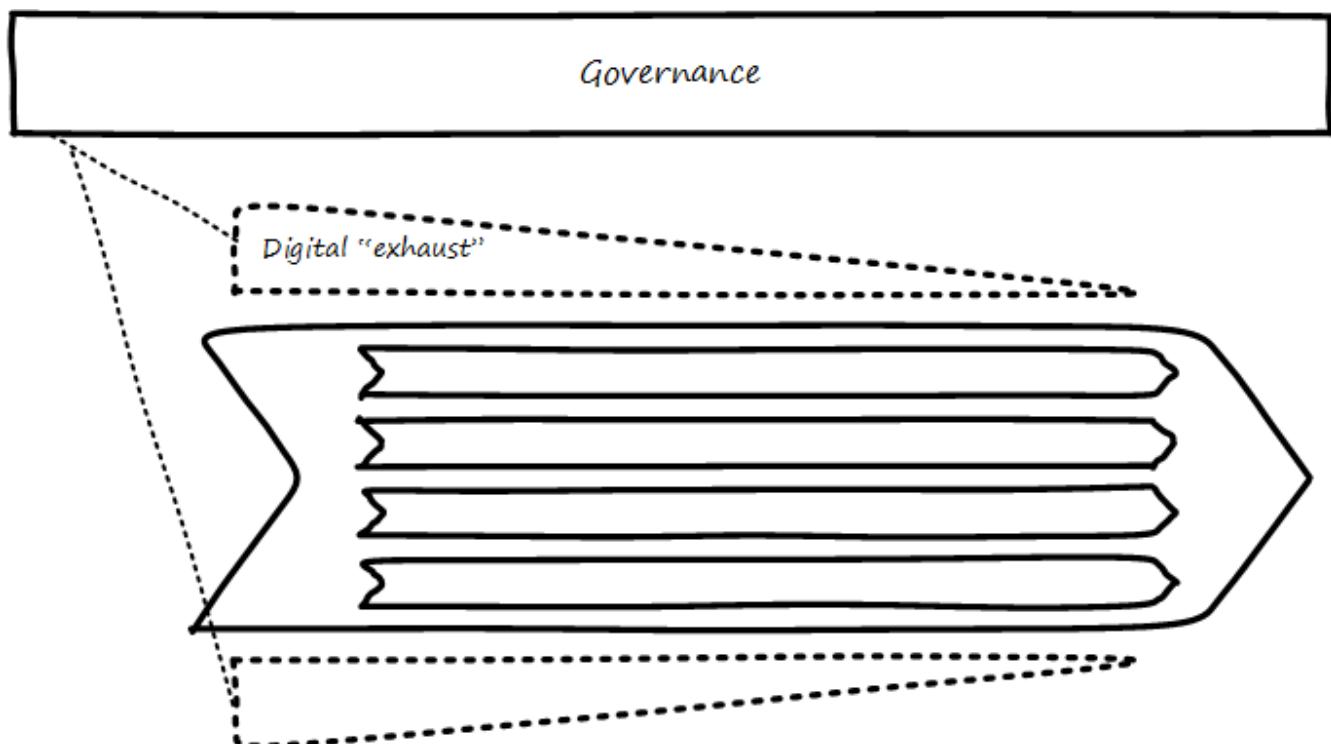
These systems constitute a core digital pipeline, one that can be viewed as an engine producing digital exhaust.

This is in contrast to fragmented, poorly-automated pipelines, or organizations with little concept of pipeline at all. Such organizations wind up relying on [secondary artifacts](#) and manual processes to deliver digital value:



The above diagram represents fragmented delivery pipelines, with many manual processes and secondary artifacts (waterfall stage approvals, designs, plans, manual ticketing, and so forth). Much IT governance assumes this model, and also assumes that governance must often rely on aggregating and monitoring the secondary artifacts.

With a rationalized continuous delivery pipeline, governance increasingly can focus on monitoring the digital exhaust:



What can we monitor with digital exhaust for the purposes of governance?

- Development team progress against backlog

- Configuration management
- Conformance to architectural standards (through inspection of source and package managers, code static analysis, and other techniques)
- Complexity and technical debt
- Performance and resource consumption of services
- Performance of standards against automated hardening activities (e.g. [Simian Army](#))

As noted above, certain governance objectives may require the pipeline itself to be adapted, e.g. the addition of static code analysis, or implementation of hardening tools such as Simian Army.

Additional automation

The DevOps Audit Toolkit

The DevOps Audit Toolkit provides an audit perspective on pipeline automation [\[DeLuccia2015\]](#). This report provides an important set of examples demonstrating how modern DevOps toolchain automation can fulfill audit objectives as well or better than "traditional" approaches. This includes a discussion of alternate approaches to the traditional control of "separation of duties" for building and deploying code. These approaches include automated code analysis and peer review as a required control.

There are a variety of ways the IT pipeline can be automated. The Calavera simulation [\[Betz2015\]](#) shows a simplified end to end approach. Many additional components are seen in real-world pipelines:

- Static code analyzers
- Automated user interface (UI) testing
- Load testing
- More sophisticated continuous deployment infrastructure

and much more.

Additionally, there may still be a need for systems that are secondary to the core pipeline.

- Service or product portfolio
- Workflow and kanban-based systems (one notable example is workflow to ensure peer review of code)
- Document management

There may also be a risk repository, if the case can't be made to track risks using some other system. The important thing to remember when automating risk management is that risks are always with respect to some **thing**.

A risk repository needs to be integrated with subject inventories, such as the service portfolio and relevant source repositories and entries in the package manager. Otherwise, risk management will

remain an inefficient, highly manual process.

What are the things that may present risks?

- Products/services
 - Their ongoing delivery
 - Their changes & transformations (Releases)
 - Their revenues
- Customers and their data
- Employees and their positions
- Assets
- Vendors
- Other critical information

The Netflix Simian Army

...just designing a fault tolerant architecture is not enough. We have to constantly test our ability to actually survive these "once in a blue moon" failures.

Imagine getting a flat tire. Even if you have a spare tire in your trunk, do you know if it is inflated? Do you have the tools to change it? And, most importantly, do you remember how to do it right? One way to make sure you can deal with a flat tire on the freeway, in the rain, in the middle of the night is to poke a hole in your tire once a week in your driveway on a Sunday afternoon and go through the drill of replacing it. This is expensive and time-consuming in the real world, but can be (almost) free and automated in the cloud.

This was our philosophy when we built Chaos Monkey, a tool that randomly disables our production instances to make sure we can survive this common type of failure without any customer impact. The name comes from the idea of unleashing a wild monkey with a weapon in your data center (or cloud region) to randomly shoot down instances and chew through cables—all the while we continue serving our customers without interruption. By running Chaos Monkey in the middle of a business day, in a carefully monitored environment with engineers standing by to address any problems, we can still learn the lessons about the weaknesses of our system, and build automatic recovery mechanisms to deal with them.

— Izrailevsky and Tseitlin, Netflix Tech Blog

The Netflix Simian Army is a collection of resiliency tools developed by the online video-streaming service Netflix. It represents a significant advancement in digital risk management, as previous control approaches too often were limited by poor scalability or human failure (e.g. forgetfulness or negligence in following manual process steps).

Chaos Monkey is one of a number of tools developed to continually "harden" the Netflix system, including:

- Latency Monkey—introduces arbitrary network delays
- Conformity Monkey—checks for consistency with architectural standards, and shuts down non-conforming instances
- Doctor Monkey—checks for longer-term evidence of instance degradation
- Janitor Monkey—checks for and destroys unused running capacity
- Security Monkey—an extension of Conformity Monkey, checks for correct security configuration

- 10-18 Monkey—checks internationalization
- Finally, Chaos Gorilla simulates the outage of an entire Amazon availability zone

On the whole, the Simian Army behaves much as antibodies do in an organic system. One notable characteristic is that the monkeys as described do not generate a report (a [secondary artifact](#)) for manual followup. They simply shut down the offending resources.

Such direct action may not be possible in many environments, but represents an ideal to work toward. It keeps the security and risk work "front and center" within the mainstream of the digital pipeline, rather than relegating it to the bothersome "additional work" it can so easily be seen as.

Topics

Collaborative

*20160713 Status: Like most topics sections, this one will remain rough until 2nd draft.
Suggestions appreciated.*

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

===== Antipatterns

===== Use of governance concerns to further agendas (Phoenix project John)

"segregation of duties" as positional defense

===== Shadow and "rogue" IT and organizational structure as control - end of CIO

===== Culture & GRC?

===== Implementation

Benchmarking

Trusted supply chain

Promise Theory

Cyberlaw (more discussion of legal in next chapter)

Hotline / whistleblowing
Social media

Industry verticals

More on cybernetics (Wiener)

Weill approach - governance as decision rights - contrast w/Kanban, ongoing iterative self-organization -

===== Projects and governance

===== Architecture and governance

Technical policies - foreshadow Chapter 12

Affect E, E, R (draw here or later)

Reality on tech product: successful digital firms strongly govern tech portfolio, do not allow experimentation that could expand base tech footprint. See chapter 12.

Globalization: Whether intentional or accidental, globalization brings business operations into foreign markets with new oversight requirements. These requirements are unique to the region and may involve controls and safeguards not necessary in other regions. <<DeLuccia2008>> p 14

If "corporate governance" is "The system by which organizations are directed and controlled," then it is reasonable that "corporate governance of IT" is "The system by which the *current and future use of IT is directed and controlled*." This is how ISO 38500, the IT Governance standard, defines it <<ISO2008>>.

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

focus on intent of controls: As Jez Humble et al note <<Humble2013>>, "In many cases, onerous rules and GRC processes that are put in place are simply somebody's interpretation of what is required, not mandated by the regulation in question.

Conclusion

Governance and its related topics are a broad and complex domain, one that represents a career path as you gain experience and seniority in your organization. It is critical to understand the motivations and objectives of governance-driven initiatives. While it can be tempting to dismiss governance-related activities as "bureaucracy," hopefully after this chapter you have a more complete understanding, and will be able to engage with governance in a constructive manner.

The following principles may be useful:

- Treat governance objectives, as much as possible, as simply a form of digital requirement to be handled through the delivery pipeline
- Consider the new classes of risk to digital effectiveness and innovation: overburden, multi-tasking, overloaded queues, and toxic culture
- Automate governance objectives throughout the pipeline. Avoid manual processing wherever possible.
- Understand that poorly conceived governance strategies and tactics can themselves constitute a risk to effective digital delivery.

Discussion questions

1. Research further and discuss one of these [IT disasters](#).
2. Propose how the disaster could have been avoided or mitigated using the governance concepts presented in this chapter.
3. Does the term "governance" bother you? Why or why not? What about the term "control"?
4. Debate, as a team, the proposition that "There is no realistic way to distinguish between governance and management." One side of the team to argue for, the other to argue against. Present your best arguments to the rest of the class.

Research & practice

- Beyond the brief discussion in this chapter, research how venture capital firms govern their

portfolios of startup investments. Compare and contrast to public company governance.

- Find a copy of the classic William Gibson short story, "Burning Chrome." Write an analysis of it using the concepts described in this chapter.
- Find a brief discussion of the human immune system, and compare and contrast it with the Netflix Simian Army.
- Compare some well known technologies you use against the NIST National Vulnerability Database. Are there any issues? How would you fix them?
- Using infrastructure as code, develop a demonstration environment employing some member of the [Simian Army](#).

Further reading

Books

- Shon Harris, [CISSP All-in-One Exam Guide](#)
- Robert Moeller, [Executive's Guide to IT Governance: Improving Systems Processes with Service Management, COBIT, and ITIL](#)
- ISACA, [COBIT 5 Framework](#)
- Gene Kim et al, [The Phoenix Project](#)
- Scott Ambler and Mark Lines, [Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise](#), Chapter 20: Governing Disciplined Agile Teams

Articles

- Wikipedia on [Corporate governance](#)
- [The top 10 IT disasters of all time](#)
- <https://technet.microsoft.com/en-us/magazine/2006.09.businessofit.aspx>
- <http://searchdatamanagement.techtarget.com/definition/compliance>
- <http://continuousdelivery.com/2013/08/risk-management-theatre/>
- [Did you wake up to an alert about the Java deserialization vulnerability](#)
- [Keeping the Auditor Away: DevOps Audit Compliance Case Studies](#)
- [Technology in Risk Management](#)

Videos

- Brian Barnier, [What Works When Monitoring Risk](#)

Professional

- [COSO \(Treadway Commission\) Internal Control — Integrated Framework \(Executive Summary\)](#)
- [ISACA - producer of COBIT](#)
- [Compliance and Ethics Association](#)

Chapter 11: Enterprise Information Management

Collaborative

Status: Chapter 11 is in first draft as of 5 September 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Introduction to Chapter 11

Information? That's nothing new. Like many other topics, we start by noting that you've been managing information in some way since the earliest days of your organization.

Perhaps your company started by offering a social media-oriented service. You needed to track users and their activity. Or perhaps you started off with a focus on data science and analytics; your first product was based on harvesting insights from large data sets. Just as we haven't in this book talked too much about the specifics of programming languages or technical architectures, we also leave the more technical aspects of data science and analytics to other textbooks and courses.

However, **it is at the largest scale that we see the establishment of formal governance, control, and management techniques for information as a distinct and coherent problem domain.** Before, you might have been interested in "customer data" or "sales data" or "supply chain data." Now, you are concerned with **data** in general, and how to manage it, wherever and whatever it is.

You're also concerned with information, knowledge, records, reports, and a wide variety of other overlapping, and often difficult to define, concepts.

Information is the lifeblood of business. Simple records are the foundation of survival: lose your customer's order, and you won't be in business long. Beyond such basics, the insights you can gain from your data are increasingly the best competitive advantage you have. Whether or not you started your company with a focus on data science, you probably have data scientists now.

The sheer scale of the data you have is starting to become unmanageable. It costs money to store, and how much do you need to store? Your lawyers have started questioning whether its truly necessary to keep certain data indefinitely; they seem to think there is risk there.

You're also periodically confronted with seemingly wasteful duplication of data - isn't it possible to be a bit more efficient? Quality and security concerns are also increasing. You've seen competitors go out of business due to cyberattacks, and you've also seen that pervasive data quality issues can be a serious threat to your operations and customer relationships.

Yet, when you have tried to manage for more efficient or better-governed uses of data, you've not been happy with the resulting cost and bureaucracy. Not to mention complaints of development delays and loss of business agility. It's an ongoing challenge, and as best you can see, no-one anywhere has it entirely figured out.

IMPORTANT

As with other chapters in the later part of this book, we are going to introduce this topic "on its own terms." We will then add additional context and critique in subsequent sections.

Chapter 11 outline

- Information and value
 - The origins of digital information
 - The measurable value of information
 - Information, efficiency, and effectiveness
- Data management basics
 - The importance of context
 - Data management and the DMBOK
 - Data architecture and development
- Towards enterprise information management
 - Advanced data management
 - Enterprise records management
 - Data Governance
- Analytics
 - Analytics in context
 - Data warehousing and business intelligence
- Agile information management
 - Software versus data
 - Next generation practices in information management
- Information management topics
 - Social, Mobile, Analytics, and Cloud
 - Big data
 - Managing the information of digital delivery
 - Additional topics

- Conclusion

Information and value

...life and reality are at bottom amorphous, disordered, contradictory, inconsistent, non-rational, and non-objective. Science and much of Western philosophy have in the past presented us with the illusion that things are otherwise. Rational views of the universe are idealized models that only approximate reality. The approximations are useful. The models are successful often enough in predicting the behavior of things that they provide a useful foundation for science and technology. But they are ultimately only approximations of reality...

— William Kent, Data and Reality

No enterprise can be effective without high quality data.

— The Data Management Body of Knowledge

The two quotes above span the challenge of information management. Fundamentally, it is an illusion. We store data and records that are *representations*. They are merely partial reflections of reality, ambiguous and requiring interpretation.

And yet, these poor shadows of the real world are the basis for all digital business. Despite their limitations, the human race has been representing and recording information in myriad ways for thousands of years.

Suppose someone hands you a piece of paper with the following:



Figure 6. Hashmarks

If you are of European descent, you might perceive this as the number twelve, or a count of twelve "somethings." What "somethings"? You don't know.



Figure 7. How many trees? [17: Image credit <https://www.flickr.com/photos/bgreenlee/1389858104/>, commercial use permitted]

Someone might tell you that this is the number of trees on your neighbor's farm. That makes all the difference. But wait - you know for a fact that there are many more trees than 12! You inquire further and find out it is actually the number of producing apple trees, a very different concept.

But the number still isn't right; the person who counted the trees didn't know the proper boundaries of the farm, and counted some that didn't belong to your neighbor.

So what does this piece of paper represent?

The origins of digital information

Writing provides a way of extending human memory by imprinting information into media less fickle than the human brain.

— AncientScripts.com

A message to mapmakers: Highways are not painted red, rivers don't have county lines running down the middle, and you can't see contour lines on a mountain.

— William Kent, Data and Reality

We've talked of **representation** previously, in the context of task management. Common representation is essential to achieving **common ground**, when human interactions are time or space shifted. Understanding representation is fundamental to understanding information.

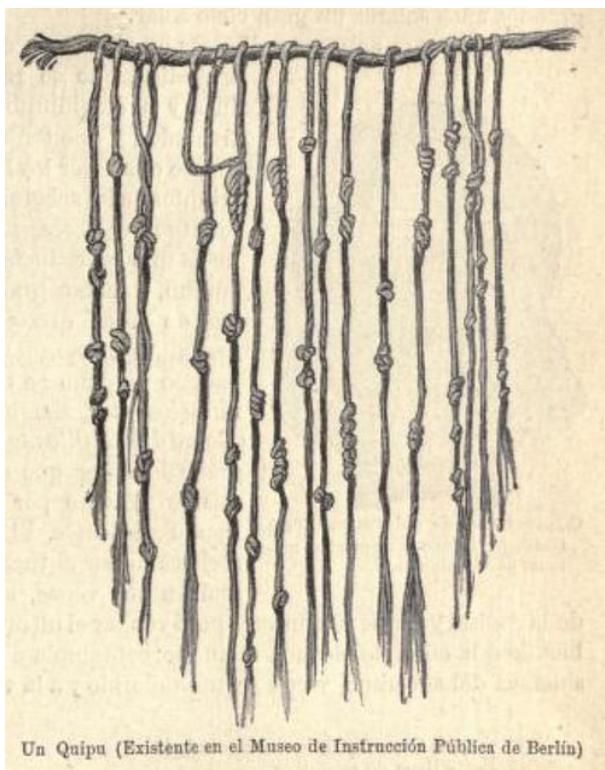


Figure 8. Sumerian cuneiform [18: Image credit <https://www.flickr.com/photos/auxesis/3920533954>, commercial use permitted]

Humans have been representing information since at least the creation of writing. As early as 3000 BCE, the ancient Sumerians used cuneiform to record—among other things—how much beer people got. (The more things change, the more they stay the same.)

Cuneiform was created by pressing wedge-shaped sticks into wet clay. A particular impression or set of impressions corresponded to a citizen, or how much grain they grew, or how much beer they received.

Certainly, the symbols below are not the same as the beer, or the grain, or the long-dead Sumerian. This may seem obvious, but it can be tempting to confuse the **representation** with the **reality**. This is known as the *reification fallacy*.



Un Quipu (Existente en el Museo de Instrucción Pública de Berlín)

Figure 9. Inca quipu [19: Image credit <https://www.flickr.com/photos/136041510@N05/24338977000>, commercial use permitted]

At around the same time as the Sumerians, the Inca empire in what is now Peru [?] also started to represent information. They used a radically different approach from the Sumerians: carefully

knotted ropes:

We will talk more about **how** information is represented. But first we need to understand **why**.

NOTE

The website ancientscripts.com has a wealth of comprehensive information on the history of writing (and therefore, information management) from its earliest techniques.

The measurable value of information

Measurement: A quantitatively expressed reduction of uncertainty based on one or more observations.

— Doug Hubbard, How to Measure Anything

Let's go back to our tree counting problem. Why do we want to count our apple trees?

We want to count them because we seek a higher level of confidence in our understanding of our assets. A quick visual scan of the trees might over or under estimate their quantity, and therefore the value of the land they are on.

We might count the neighbors' apple trees because we want to buy their orchard. We might count our sheep because we are thinking about going into business with our neighbor and we need to know what each side is bringing to the effort. All kinds of critical life and business decisions depend on having confidence in information.

In fact, there is often a cost to **not** having the information one needs. Suppose you are considering purchasing your neighbor's farm. You believe it has more trees on it, than it does in reality. As a result you pay more than the land is worth, for example you pay 250 gold pieces when it was worth only 200. In this case, we can argue that the value of better information would have been 50 gold pieces.

If collecting the needed information (say, by bribing the local bartender who knows all the gossip) is 25 gold pieces, there would have been a 100% return in the investment of collecting the information.

How does this relate to data and information management? Let's consider another example.

Suppose you run the gas station in a small town. A very small town, so small that you know all your customers by name. Sometimes you extend them credit, but this is done on an honor system and neither you nor your customer writes anything down. (Yes, this is unrealistic, even for the smallest merchant, but consider it a thought experiment.)



Figure 10. What do they owe you? [20: Image credit

<https://www.flickr.com/photos/stevensnodgrass/6625148339>, commercial use permitted]

As the town grows, and your customers increase, you grow increasingly uncertain as to whether they all are paying you what they owe. You purchase a notebook and a pen (today's equivalent of clay tablets or knotted ropes) and start writing down the amounts owed you.

There is cost to this: both the paper and pen cost you money, as well as the time you are now spending on record keeping and not pumping gas or operating the cash register. The benefit however is that you are now certain that you are collecting money owed you. You might notice that your profit margins have gone up by 0.5%; that is the value of better record keeping.

There are opportunity costs. Perhaps you have to hire an attendant sooner. But there has clearly been return on your investment in better information management.

You expand further, and see that your paper-based system has gaps. The volume of transactions is now such that you again have uncertainty about whether you are receiving all the money owed you. You decide to automate, to invest in an electronic record keeping system. This might cost tens of thousands of dollars in one time and ongoing costs, but again you see that your profit margins on balance improve. Again, an increase in your certainty of information resulted in value. (Of course in reality, you would stop extending credit personally and require your customers to use credit cards.)

The point of these examples is to emphasize that all information management can be understood in terms of the Doug Hubbard quote at the start of this chapter section: information is a reduction in uncertainty, whether regarding apple trees on a tract of land, or accounts receivable. And we can and should quantify the value of having the information, versus the cost of capturing and maintaining it.

Doug Hubbard, in the classic *How to Measure Anything* [Hubbard2010], asks the following questions when measurement is proposed (page 47):

1. What is the decision this measurement is supposed to support?
2. What is the definition of the thing being measured in terms of observable consequences?
3. How, exactly, does this thing matter to the decision being asked?

4. How much do you know about it now (i.e., what is your current level of uncertainty)?
5. What is the value of additional information?

As he states, "All measurements that have value must reduce the uncertainty of some that affects some decision with economic consequences." While Hubbard is proposing these questions in the context of particular analysis initiatives, they are also excellent questions to ask of **any** operational information management.

Information management, in the context of digital systems, adds value through improving efficiency, effectiveness, and optimizing risk (our three primary categories of value.) Since digital systems started off primarily as efficiency aids, we will discuss efficiency first.

Information, efficiency, and effectiveness

We have periodically discussed historical aspects of computing and digital systems, but not yet covered some of the fundamental motivations for their invention and development.

As technology progressed through the late 19th and early 20th centuries, applied mathematics became increasingly important in a wide variety of areas such as:

- Ballistics (e.g. artillery) calculations
- Cryptography
- Atomic weapons
- Aeronautics
- Stress and load calculations

Calculations were performed by "computers." These were not automated devices, but rather people, often women, tasked with endless, repetitive operation of simple adding machines, by which they manually executed tedious calculations to compile (for example) tables of trigonometric angles.

It was apparent at least since the mid-19th century that it would be possible to automate such calculation. In fact, mathematical devices had long existed, for example the abacus, Napiers' Bones, and the slide rule. But such devices had many limitations. The vision of automating digital calculations first came to practical realization through the work of Charles Babbage and Ada Lovelace.

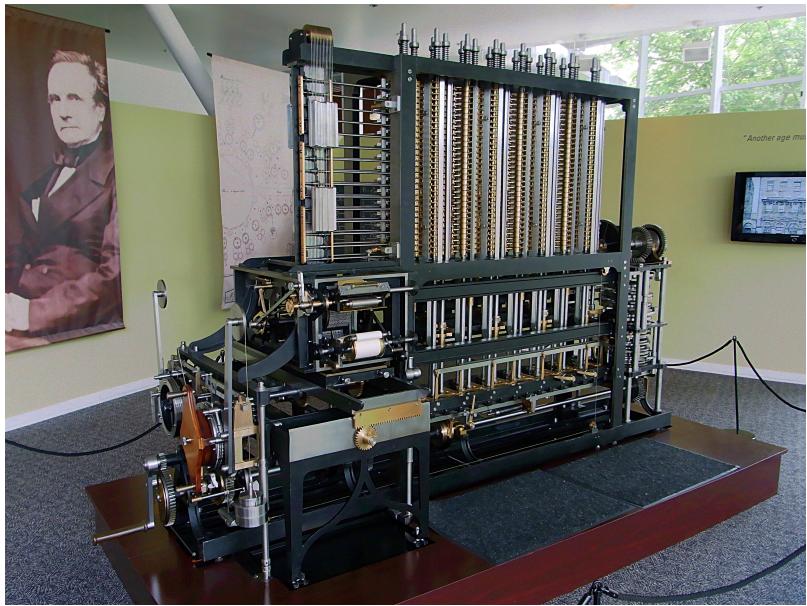


Figure 11. Babbage Difference Engine [21: Image credit

<https://www.flickr.com/photos/tolomea/4847227935>, commercial use permitted]

Charles Babbage, Ada Lovelace, and the Engines

Charles Babbage (1791-1871) was a 19th century intellectual and inventor who devised the first practical apparatus for performing general purpose calculations. The ability to add sums and multiply limited numbers had long been automated [examples] but Babbage's "Difference Engine" was capable of more complex operations (calculating polynomials).

Babbage's Difference Engine (pictured in this chapter section) was not a general purpose computer, but as he was designing it he started to plan a more ambitious "Analytical Engine" which would have been a general purpose computer. However, designs for the Analytical Engine were never completed. Nor was even the Difference Engine successfully built during Babbage's lifetime, although its design was solid enough that at least one fully functioning example has been constructed in recent years.

Ada Lovelace's (1815-1852) contribution was equally remarkable. She is generally credited with inventing the "algorithm," realizing that computation was a repeatable process that could be precisely specified, independently of its inputs and outputs - a signature intellectual moment for the human race.

To characterize these two remarkable historical figures affectionately in terms of this book, Babbage was the first infrastructure engineer, and Lovelace the first applications developer.

For a fun, accessible, and charming exploration of their activities, read *The Thrilling Adventures of Lovelace and Babbage: The (Mostly) True Story of the First Computer* [Padua2015].

After Babbage, the development of automated computation encountered a hiatus. Purely mechanical approaches based on gears and rods could not scale, and the manufacturing technology of Babbage's day was inadequate to his visions - the necessary precision and power could not be achieved for implementing a general purpose computer using his legions of gears, cams, and drive

shafts.

However, mathematicians continued to explore these areas, culminating in the work of Alan Turing who established both the potential and the limits of computing, initially as a by-product of investigations into certain mathematical problems of interest at the time.

Around the same time, the legendary telecommunications engineer Claude Shannon had developed essential underpinning engineering in expressing Boolean logic in terms of electronic circuits and rigorous mathematical theory describing the fundamental characteristics and limitations of information transmission (e.g. the physical limits of copying one bit of data from one location from another) [Shannon1949].

Advances in materials and manufacturing technique resulted in the vacuum tube, ideally suited to the combination of Shannon digital logic with Turing's theories of computation, and thus the computer was born. It's generally recognized that the first practical general purpose computer was developed by the German Konrad Zuse.

Turing and a fast-growing cohort of peers driven by (among other things) the necessities of World War II developed both theory and the necessary practical understandings to automate digital computation. The earliest machines were used to calculate artillery trajectories. During World War II, mathematicians and physicists such as John von Neumann recognized the potential of automated computation, and so computers were soon also used to simulate nuclear explosions. This was a critical leap beyond the limits of manual "computers" pounding out calculations on adding machines.

The business world was also attentive to the development of computers. Punched cards had been used for storing data for decades preceding the invention of automated computers. Record keeping at scale has always been challenging - the number of Sumerian clay tablets still in existence testifies to that! Industrial-era banks, insurers, and counting-houses managed massive repositories of paper journals and files, at great cost. A new form of professional, the "white collar worker" emerged.

Any means of reducing the cost of this record keeping was of keen interest. Paper files were replaced by punched cards. Laborious manual tabulation was replaced by mechanical and electro-mechanical techniques, that could for example calculate sums and averages across a stack of punched cards, or sort through the stack, compare it against a control card and sort the cards accordingly.

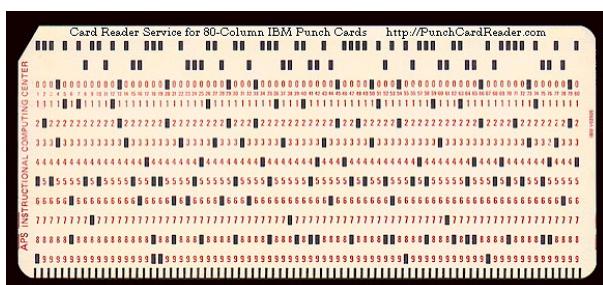


Figure 12. Punchcard [22: Image credit <https://www.flickr.com/photos/129953653@N05/16540457804>, commercial use permitted]



Figure 13. Card Sorter [23: Image credit <https://www.flickr.com/photos/pargon/2444932424>, commercial use permitted]

During World War II, many business professionals found themselves in the military, and some encountered the new electronic computers being used to calculate artillery trajectories or decrypt enemy messages. Edmund Berkeley, the first secretary of the Association for Computing Machinery, was one such professional who grasped the potential of the new technology [Akera2007]. After the war, Berkeley advocated for the use of these machines to the leadership of the Prudential insurance company, while others did the same in the United Kingdom.

What is the legacy of Babbage and Lovelace and their successors in terms of today's digital economy? The reality is that digital value for the first 60 years of fully automated computing systems was primarily in service of efficiency. In particular, record keeping was a key concern.

Business computing (as distinct from research computing) had one primary driver: efficiency. Existing business models were simply accelerated with the computer. 300 clerks could be replaced by a \$10 million machine and a staff of 20 to run it (at least, that was what the sales representative promised.) And while there were notable failures, the value proposition held up such that computer technology continued to attract the necessary R&D spending, and new generations of computers started to march forth from the laboratories of Univac, IBM, Hewlett-Packard, Control Data, Burroughs, and others.

Efficiency ultimately is only part of business value. Digital technology relentlessly wrings out manual effort and this process of automation is now so familiar and widespread that it is not necessarily a competitive advantage. Harvard Business Review editor Nicholas Carr became aware of this in 2003. He wrote a widely discussed article, "IT Doesn't Matter," in which he argued that "When a resource becomes essential to competition but inconsequential to strategy, the risks it creates become more important than the advantages it provides" [Carr2003].

Carr compared information technology to electricity, noting that companies in the early 20th century had Vice Presidents of Electricity and predicting the same for Chief Information Officers.

Carr's article provoked much discussion at the time, and continues to be influential. It remains important and insightful. Certainly, to the extent IT's value proposition is coupled only to efficiency (e.g. automating clerical operations), IT is probably less important to strategy.

But as we have discussed throughout this book, IT is permeating business operations, and the traditional CIO role is in question as mainstream product development becomes increasingly digital. The value of correctly and carefully applied digital technology is more variable than the value of electricity. At this 2016 writing, the four largest companies by market capitalization (Apple, Google, Facebook, and Microsoft) are digital firms, based on digital products, the result of digital strategies based on correct understanding and creative application of digital resources.

In this world, information enables effectiveness as much or more as efficiency.

Data management basics

The importance of context

Information management as we will discuss in the rest of this chapter arises from the large scale absorption of data into highly efficient, miniaturized, automated digital infrastructures with capacity orders of magnitude greater than anything previously known.

But cuneiform and quipu, hash marks on paper, financial ledgers, punched cards, vacuum tubes, transistors, and hard disks represent a continuum, not a disconnected list. Whether we are looking at a scratch on a clay tablet, or the magnetic state of some atoms in a solid state drive, there is one essential question:

What do we mean by that?

Consider the state of those atoms on a solid state drive. They represent the numbers 547. But without context, that number is meaningless. It could be:

- The numeric portion of a street address
- A piece of a taxpayer identification number
- The balance on a bank account
- A piece of the data uniquely identifying DNA in a crime

The state of this data may have significant consequences. A destination address might be wrong, a tax return mis-identified. A credit card might be accepted or declined. A mortgage might be approved or denied. Or the full force of the law may be imposed on an offender, including criminal penalties resulting from a conviction on the evidence stored in the computer.

The COBIT Enabling Information guide [\[ISACA2012b\]](#) proposes a layered approach to this problem:

Layer	Implication
Physical	The media (paper, electronic) storing the data
Empirical	The layer that observes the signals from the physical, and distinguishes signal from noise
Syntactic	The layer that encodes the data into symbols (e.g. ASCII)

Layer	Implication
Semantic	The layer providing the rules for constructing meaning from syntactical elements
Pragmatic	The layer providing larger, linguistic structuring
Social	The layer that provides the context and ultimately consequence of the data (e.g. legal, financial, entertainment)

Without all these layers, the magnetic state of those atoms is irrelevant.

The physical, empirical, and syntactic layers - (hardware and lowest level software) are in general out of scope for this book. They are the concern of broad and deep fields of theory, research, development, market activity, and standards. (Chapter 2 on infrastructure is the most closely related).

A similar but simpler hierarchy is

- Data
- Information
- Knowledge

Data is the context-less raw material.

Information is data + context, which makes it meaningful and actionable

Knowledge is the situational awareness to make use of information

Semantic, pragmatic, and social concerns (information and knowledge) are fundamental to this book and chapter. At digital scale - terabytes, petabytes, exabytes - establishing the meaning and social consequence of data is a massive undertaking. Data management and records management are two practices by which such meaning is developed and managed operationally. We will start by examining data management as a practice.

Data management and the DMBOK

Data management is a long established practice in larger IT organizations. As a profession, it is represented by the Data Management Association (DAMA). DAMA developed and supports the Data Management Body of Knowledge (DMBOK), which is a primary influence on this chapter section.

The Data Management Body of Knowledge

The Data Management Body of Knowledge [DAMA2009] is similar to other frameworks presented in this book (e.g. ITIL, COBIT, and PMBOK.) It includes ten major functions:

- Data Governance
- Data Architecture Management

- Data Development
- Data Operations Management
- Data Security Management
- Reference and Master Data Management
- Data Warehousing and Business Intelligence Management
- Document and Content Management
- Meta-data Management
- Data Quality Management

Attentive readers will notice some commonalities with general areas covered in this book: Governance, Architecture, Operations, and Security in particular. Data at scale is a significant problem area and so the DMBOK provides a data-specific interpretation of these broader concerns, as well as more specialized topics.

We won't go through each of the DMBOK functions in order, but we will be addressing most of them throughout this chapter.

Data architecture and development

Data and process

In order to understand data, we must understand how it is being used. We covered process management in Chapter 9. Data is often contrasted with process: processes take data inputs and produce data outputs. The fundamental difference between the two can be seen in the core computer science concepts of algorithms (process) and data structures.

Data emerges, almost unavoidably, when processes are conceived and implemented. A process such as "Hire Employee" implies that there is an employee, and also a concept of "hire" with associated date and other circumstances.

This may seem obvious, but data structures are surprisingly challenging to develop and gain consensus on.

The ontology problem

The boundaries of an entity are arbitrary, our selection of entity types is arbitrary, the distinction between entities, attributes, and relationships is arbitrary.

— Graeme Simsion, Preface to Kent's Data and Reality

Suppose you are discussing the concept of "customer" with a teammate. You seem to be having some difficulty understanding each other. (You are from Support and she is from Sales.) You begin to realize that you have two different definitions for the same word:

- You believe that "customer" means someone who has bought something
- She believes that "customer" includes sales leads

This is a classic issue in data management: when one term means two things. It can lead to serious confusion and technical difficulties, if these misunderstandings affect how systems are built and operated.

Because of this, it is critical to have rational and clear discussions about "what we mean." In a startup driven by one or two visionary founders, perhaps little or no time is needed for this. The mental model of the problem domain may be powerfully understood by the founder, who controls the key architectural decisions. In this way a startup can progress far with little formalized concern for data management.

But as a company scales, especially into multi-product operations, unspoken (tacit) understandings do not scale correspondingly. Team members will start to misunderstand each other, unless definitions are established. This may well be needed regardless of whether data is being held in a database.

The concept of a "controlled vocabulary" is therefore key to enterprise information management.

Definition: Controlled Vocabulary

"A controlled vocabulary is an information tool that contains standardized words and phrases used to refer to ideas, physical characteristics, people, places, events, subject matter, and many other concepts. Controlled vocabularies allow for the categorization, indexing, and retrieval of information." [\[Harpring2010\]](#)

In many areas of business, the industry defines the vocabulary. Retailers are clear on terms like "supplier," "cost," and "retail" (as in amount to be charged for the item; they do not favor the term "price" as it is ambiguous.) The medical profession defines "patient," "provider," and so forth.

But in more flexible spaces, where a company may be creating its own business model, defining a controlled vocabulary may be essential. We see this even in books, which adopt glossaries. Why does a book have a glossary, when dictionaries exist? Because the book is defining a controlled vocabulary. General-purpose dictionaries may list multiple meanings for the same word, or not be very precise. The author, by developing a glossary, can make the book more consistent and accurate.

There are techniques for developing controlled vocabularies in efficient and effective ways. The term "ontology engineering" is sometimes used [\[DeNicola216\]](#). While specialists may debate the boundaries, another important practice is "conceptual data modeling."

All of these concepts (controlled vocabularies, glossaries, ontologies, conceptual data models) are independent of computers. But the initial development of controlled vocabulary is the first step towards automating the information with computers.

Data modeling

An information system (e.g., database) is a model of a small, finite subset of the real world...We expect certain correspondences between constructs inside the information system and in the real world. We expect to have one record in the employee file for each person employed by the company. If an employee works in a certain department, we expect to find that department's number in that employee's record.

— William Kent, Data and Reality

Databases are the physical representation of information within computing systems. As we discussed above, the data contained within them **corresponds** to some "real world" concept we hold.

There are well known techniques for translating concepts (e.g. controlled vocabularies) into technical database structures. The best known of these is relational data modeling.

Relational data modeling is often presented as having three layers:

- Conceptual
- Logical
- Physical

The following descriptions of the layers are typical:

Conceptual	Independent of computing platform - no assumption of any database. Does include simple relationships. Does not include attributes.
Logical	Assumes a database, but not what kind. Includes more detailed relationships and attributes. Human-readable names.
Physical	Intended for a specific database platform (e.g. Oracle or MySQL). Computer-compatible names. Can be used to generate data definition scripts.

A simple conceptual model might look like this:

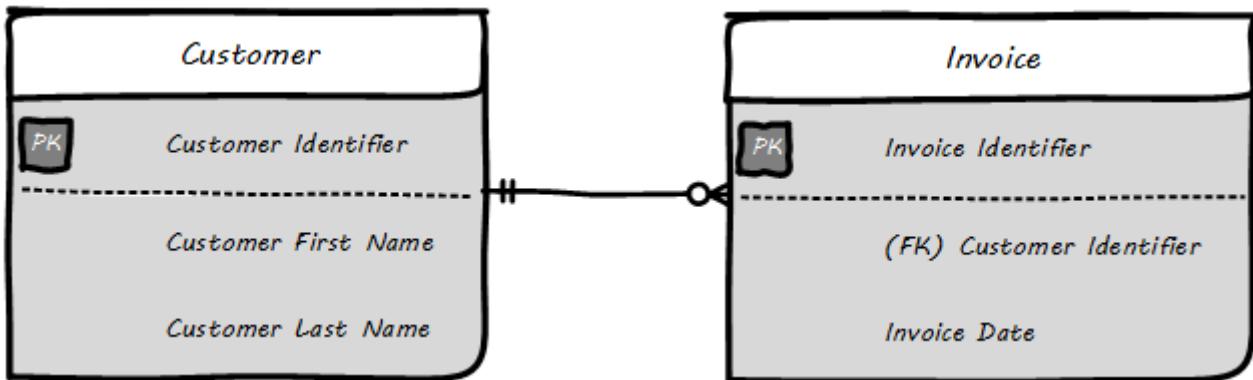


The above model might be a fragment from a sales system. It shows that there are four major *entities*:

- Customer
- Invoice

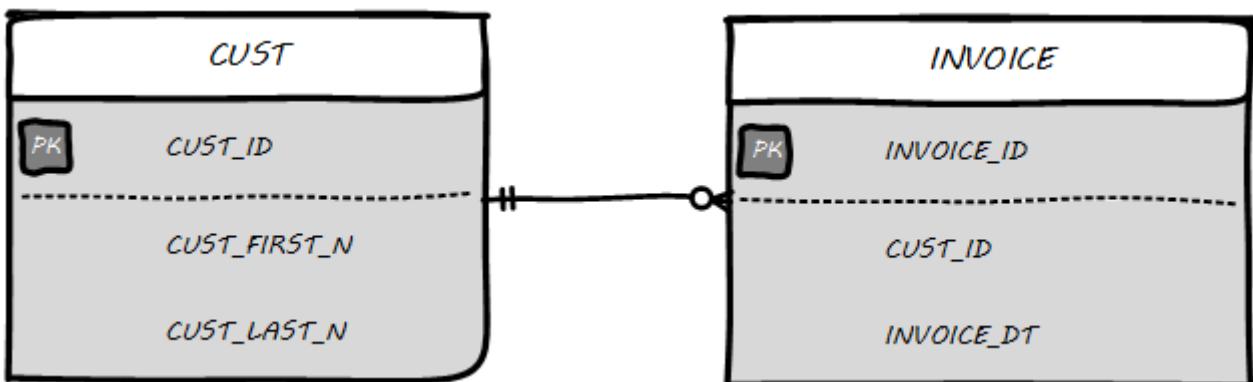
- Line Item
- Product

This might be elaborated into a logical model:



The logical model includes *attributes* (Customer First Name). The line between them has particular "adornments" representing a well known data modeling notation called "crow's foot." In this case, the notation is stipulating that one customer may have zero to many invoices, but any invoice must have one and only one customer. Notice also that the entity and attribute names are human-readable.

Then, the logical model might be transformed into physical:



The names are no longer human-readable in full, nor do they have spaces. Common data types such as "name" and "date" have been replaced with brief codes ("N" and "DT"). In this form, the physical data model can be (in theory) translated to data definition language that can result in the creation of the necessary database tables.

Database administration

Continuing from above: the data modeling work might have been performed by a data architect or analyst, or a developer. Perhaps a pictorial representation is not even created (formal data modeling as above is less likely in a startup). But at some point (assuming a relational database) the following statement will be developed:

```
CREATE TABLE SALES.CUST  
(CUST_ID NUMBER,  
CUST_FIRST_N VARCHAR2(32),  
CUST_LAST_N VARCHAR2(32))
```

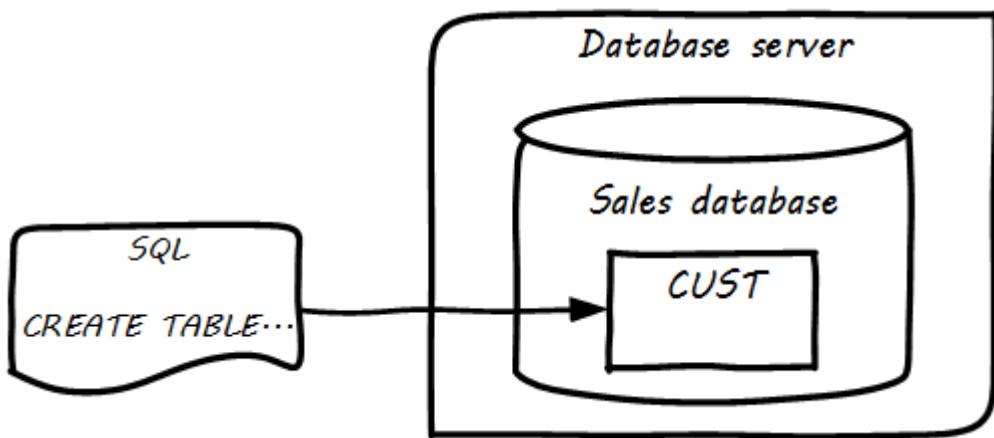
In the above SQL (Structured Query Language) statement, the Customer entity has been finally represented as a series of encoded statements an Oracle database can understand, including specification of the data types needed to contain Customer Identifier (a Number type) and the customer's first and last names (a 32-character long string field, called "VARCHAR" in Oracle).

If a database administrator (DBA) issues that statement to the Oracle database, the table will be created. Once the structure is created, it can (within limits) hold any number of customers, in terms of knowing their first and last names and an ID number, which might or might not be assigned automatically by the system. (Of course, we would want many more attributes, e.g. customer address.)

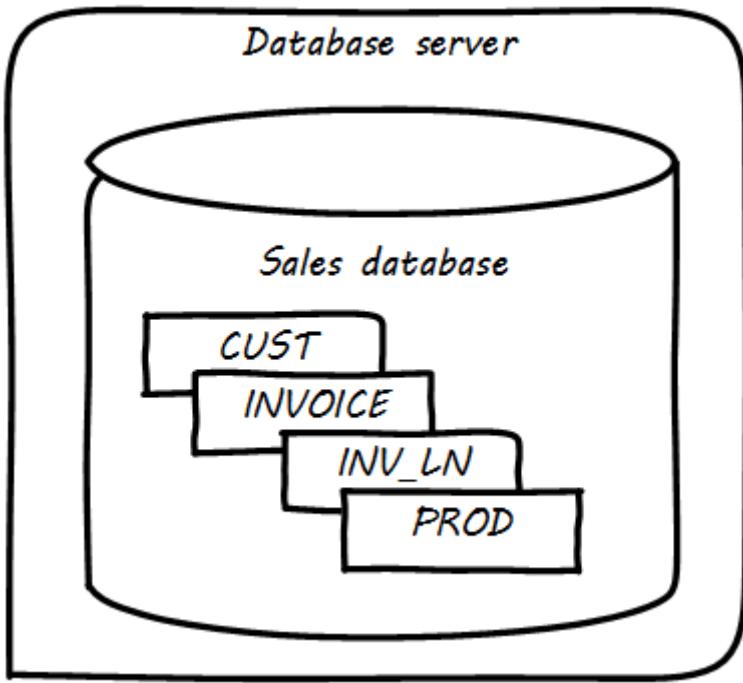
IMPORTANT

Notice that this database would only work for regions where customers have "first" and "last" names. This may not be true in all areas of the world. See [Falsehoods Programmers Believe about Names](#).

The Oracle software is installed on some node or machine, and receives the statement. The database adds the table suggested:



Further tables can easily be added in the same manner:



What is a database in this sense? The important point is that it is a common query space - you can ask for data from either the CUST or INVOICE table, or ask the database to "join" them so you can see data from both. (This is how, for example, we would report on sales by customer.)

Patterns and reference architectures

Reference architectures and design patterns are examples of approaches that are known to work for solving certain problems. In other words, they are reusable (and usually free) solutions for commonly occurring scenarios. They apply to core software development, often suggesting particular class structures. [\[Gamma1995\]](#) However, the concept can also be applied to data and system architectures, e.g. [\[Fowler2003\]](#), [\[Betz2011a\]](#). David Hay [\[Hay1996\]](#) and Len Silverston [\[Silverston2001\]](#), [\[Silverston2001a\]](#), [\[Silverston2008\]](#) have documented data models for a variety of industries.

Reference architectures also can provide guidance on data structures, as they often contain industry learnings. Examples include:

Organization	Domain	Standard(s)
Tele-Management Forum	Telecommunications	Frameworx, ETom (Enhanced Telecommunications Operating Model), NGOSS, SIDS
Association for Retail Technology Standards	Retail	ARTS model
ACORD.org	Insurance	ACORD Framework
Banking Industry Architecture Network	Banking	BIAN Service Landscape
The Open Group Exploration, Mining, Metals and Minerals Forum	Exploration, Mining, and Minerals	Exploration and Mining Business Reference Model

Organization	Domain	Standard(s)
The Open Group IT4IT Forum	Information Technology Management	IT4IT Standard

Patterns and reference architectures can accelerate understanding, but they also can over-complicate solutions. Understanding and applying them pragmatically is the challenge. Certainly, various well-known problems such as customer address management have surprising complexity, and can benefit from leveraging previous work.

Section conclusion

The above description is brief and "classic" - the techniques shown here date back decades, and there are many other ways the same problem might be represented, analyzed, and solved. But in all cases in data management, the following questions must be answered:

- What do we mean?
- How do we represent it?

The classic model shown here has solved many business problems at large scale. But there are critical limitations. Continuing to expand one "monolithic" database does not work past a certain point, but fragmenting the data into multiple independent systems and datastores also has challenges. We will discuss these further as the chapter progresses.

Towards enterprise information management

The previous chapter was necessary, but narrow. From those basic tools of defining controlled vocabularies and mapping them onto computing platforms, has come today's digital economy and its exabytes of data.

The relational database as represented in the previous chapter can scale, as a single unit, to surprising volumes and complexity. Perhaps the most sophisticated and large scale examples are seen in Enterprise Resource Planning (more detail on this in the Topics section.) An ERP system can manage the supply chain, financials, human resources, and manufacturing operations for a Fortune 50 corporation, and itself constitute terabytes and tens of thousands of tables.

However, ERP vendors do not sell solutions for leading edge digital transformation. They represent the commoditization side of the [Innovation Cycle](#). One cannot base a digital go-to-market strategy on them, as everyone has them.

Competing in the market requires systems of greater originality and flexibility, and that usually means some commitment to either developing them with internal staff or partnering closely with someone who has the necessary skills.

And as these unique systems scale up, a variety of concerns emerge, requiring specialized perspectives and practices.

Advanced data management

The previous chapter section gave us the basics of data storage. We turn to some of the emergent

practices seen as enterprise information management scales up:

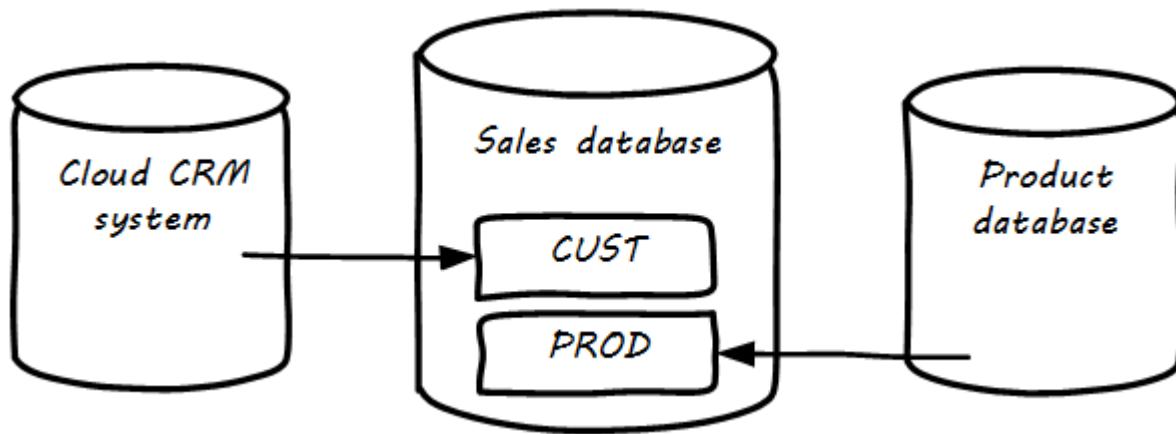
- Data integration and the "System of Record"
- Reference data management
- Data quality management

Data integration and the "System of Record"

In the last section, we analyzed a business problem as a data model and created a simple database. Notice that if data is NOT in a table in the database, the database doesn't know about it. Therefore, if there is data held in other parts of the company, it must be loaded into the database before it can be combined with that database's other data. This can also apply to data held by Cloud providers.

Let's go back to our [emergence model](#). Think about moving from one database, to two.

In the example below, the Customer Relationship Management (CRM) system is in the cloud, and data is also being imported from the product database:



The process of identifying such remote data and loading it into a database to enable work to be done is known as "integration" and is a complex domain [\[Hohpe2003\]](#). There are many ways data can be integrated, and industry views of what is "good practice" have changed over the years.

NOTE Thinking in terms of the emergence model, you have likely been integrating data in various ways for some time. However, in a large governed organization, you need to formalize your understandings and approach.

Take the above diagram and multiply it by several hundred times, and you will start to get an idea of the complexity of enterprise information management at scale in the enterprise. How do we decide what data needs to flow where? What happens if we acquire another company and can't simply move them over to our systems immediately? What department should properly own a given database? These and similar questions occupy data professionals worldwide.

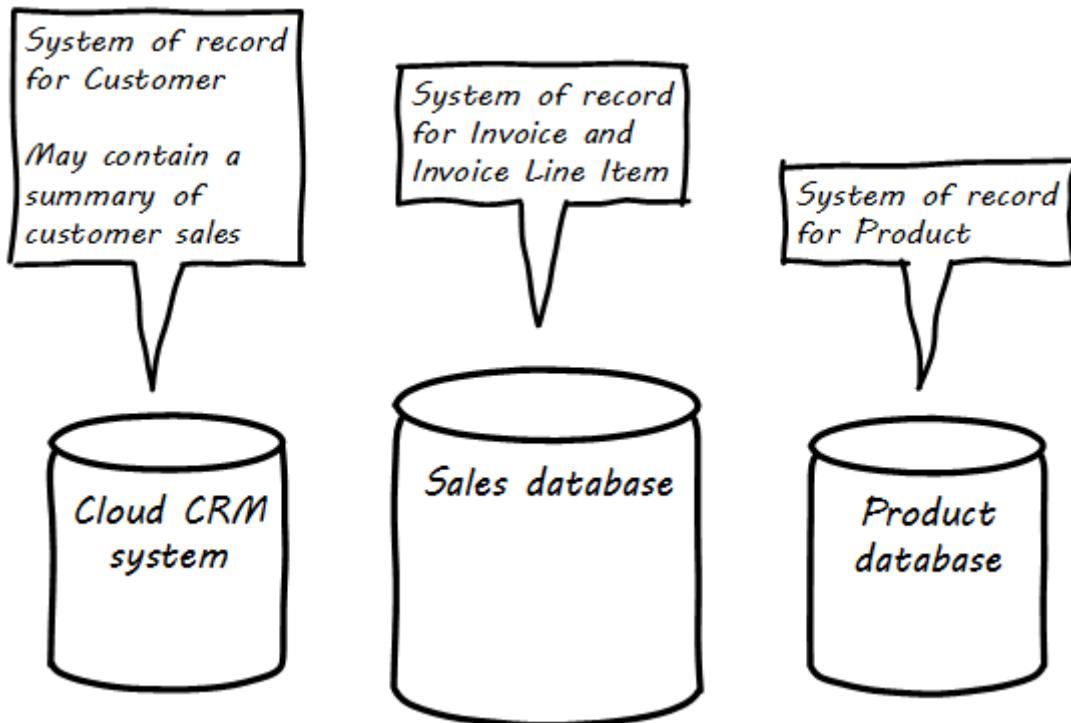
As we see from the above picture, the same data may exist in multiple systems. Understanding what system is the "master" is critical. Product data should (in general) not flow from the sales database back into the product system. But what about sales information flowing from the sales database back to the CRM system? This might be helpful, so that people using the CRM system

understand how much business a customer represents.

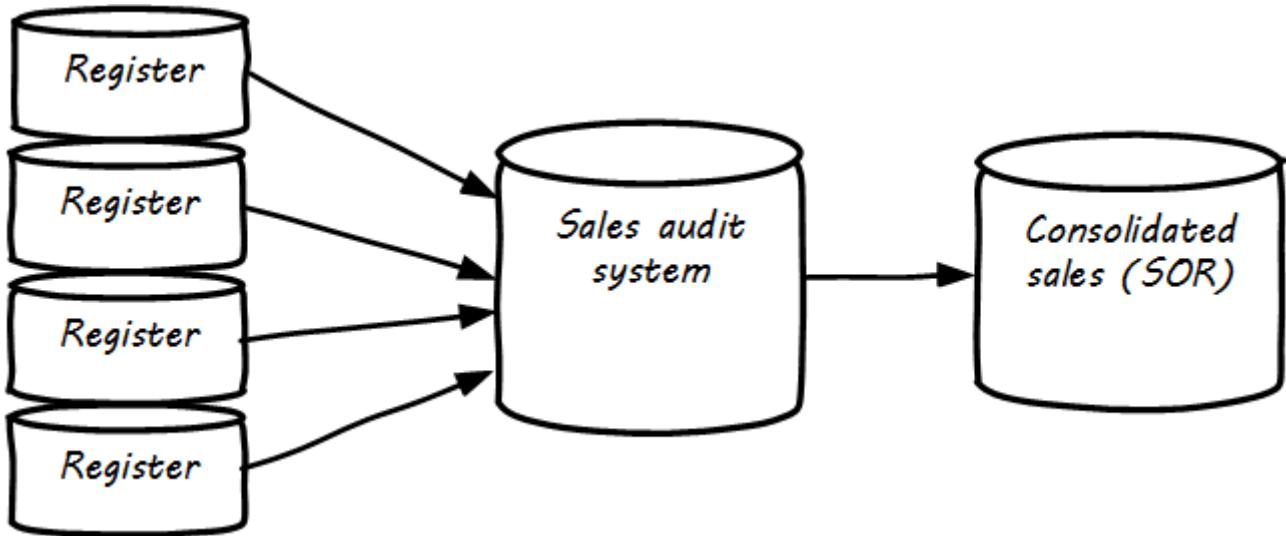
The "system of record" concept is widely used in data management, records management, and enterprise architecture to resolve these kinds of questions. It is often said that the system of record is the "master" for the data in question, and sustaining the System of Record concept may also be called Master Data Management. In general, the system of record represents data that is viewed as:

- the most complete and accurate
- authoritative, in terms of resolving questions
- suitable for legal and regulatory purposes
- the "source" for other systems to refer to

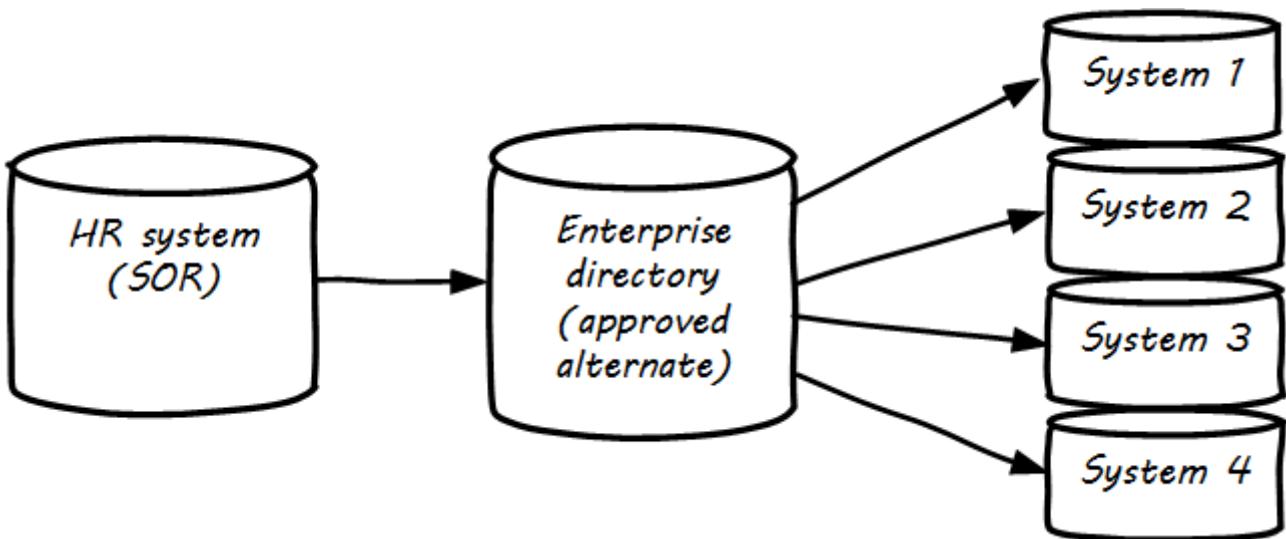
It is important to realize that the designation "system of record" is a ROLE that a given database (or system) plays with respect to SOME DATA. The "Sales" database above might be the system of record for invoices, but is not the system of record for products or customers:



The system of record (aka SOR) is often the system of origin - the system where the data is first captured - but not always. In retail, sales transactions are not considered authoritative until they pass through sales audit, and so the system of record for a transaction is not the cash register (which has a database in it):



Systems of record logically would be the first place to pull data from, but sometimes due to performance or security concerns data may be replicated into an alternate source better suited for distributing the data. A good example of this is an HR system that feeds the corporate directory; the HR system (e.g. Oracle HR, or Workday) is the actual system of record for employee names and identifiers, but most people and other systems in the company will pull the data from the directory (e.g. Microsoft Exchange):



Reference data management

Reference data is any kind of data that is used solely to categorize other data found in a database, or solely for relating data in a database to information beyond the boundaries of the enterprise.

— Malcolm Chisholm, Managing Reference Data

There are various ways to categorize data. Frequently, it is categorized by business process or functional area (Sales, Marketing, Support, etc). However, another important way to understand data is by its non-functional characteristics. Some data is transactional (e.g. Invoice and Invoice Line Item in the above example) and some is more persistent (Customer, Product).

When data is highly persistent and used for primarily for categorizing across an enterprise, we may call it "reference" data. Classic examples of reference data include:

- Geographic information (cities, states, zip codes)
- Widely used "codes" in the business (medical insurance makes extensive use of these)
- An organization's chart of accounts

Reference data is among the first data to be shared between systems. It is often the basis for dimensions in analytic processing, which we cover in the next chapter section.

Commercial data

Data is not just an internal artifact of organizations. Data can be bought and sold on the open market like any other commodity. Marketing organizations frequently acquire contact information for email and other campaigns. Here are examples of commercial data available through market sources:

Data type	Market provider
Stock prices	Bloomberg, Reuters
Credit ratings	Trans-Union, Experian
Known security issues	NIST Common Vulnerability database

Other forms include:

- Transactions of record (e.g. real estate)
- Governmental actions (these may be nominally free from the government, but frequently are resold by vendors who make the data more accessible)

For a detailed examination of the privacy issues relating to commercial data, especially when governmental data is commercialized, see [\[DHS2006\]](#).

Data quality

Human beings cannot make effective business decisions with flawed, incomplete, or misleading data.

— Danette McGilvary, Executing Data Quality Projects

We touched on quality management and [continuous improvement](#) in Chapter 9. Data is an important subject for continuous improvement approaches. Sometimes, the best way to improve a process is to examine the data it is consuming and producing, and an entire field of data quality management has arisen (a number of books are referenced at the end of this chapter).

Poor data quality costs the enterprise in many ways.

- Customer dissatisfaction ("they lost my order/reservation")
- Increased support costs (30 minutes support operator time spent solving the problem)
- Governance issues and regulatory risk ([auditors](#) and regulators often check data quality as evidence of [compliance](#) to [controls](#) and regulations)
- Operational and supply chain issues
- Poor business outcomes

Sidebar: The power of bad data

In 2015, Minneapolis-based Target Corporation lost \$7 billion when its attempt to expand into Canada failed. One primary reason for the loss was a failure of data quality in the supply chain system. As reported by *Canadian Business*:

"It didn't take long for Target to figure out the underlying cause of the breakdown: The data contained within the company's supply chain software, which governs the movement of inventory, was riddled with flaws.

"Product dimensions would be in inches, not centimeters or entered in the wrong order: width by height by length, instead of, say, length by width by height. Sometimes the wrong currency was used. Item descriptions were vague. Important information was missing. There were myriad typos. "You name it, it was wrong," says a former employee. "It was a disaster."

The consequences were serious. "...products weren't fitting into shipping containers as expected, or tariff codes were missing or incomplete. Merchandise that made it to a distribution center couldn't be processed for shipping to a store. Other items weren't able to fit properly onto store shelves. What appeared to be isolated fires quickly became a raging inferno threatening to destroy the company's supply chain."

Ultimately, lack of inventory (empty shelves in the store) was a major cause of the expansion's failure. [\[Castaldo2016\]](#)

The following activities are typically seen in data quality management (derived and paraphrased from [\[DAMA2009\]](#)):

- Identify measurable indicators of data quality
- Establish a process for acting upon those indicators (what do we do if we see bad data?)
- Actively monitor the quality
- Fix both data quality exceptions, and their reasons for occurring

Data quality indicators may be automated (e.g. reports that identify exceptions) or manual (e.g. audits of specific records and comparison against what they are supposed to represent).

It is important to track trending over time, so that the organization understands if progress is being made.

Enterprise records management

Not all enterprise information is stored in structured databases; in fact, most isn't. (We will leave aside the issues of rich content such as audio, images, and video.)

Content management is a major domain in and of itself, which shades into the general topic of knowledge management (to be covered in the Topics section). Here, we will focus on records management.

As discussed above, businesses gained efficiency through converting [paper records to digital forms](#). But we still see paper records to this day: loan applications, doctor's forms, and more. If you have a car, you likely have an official paper title to it issued by a governmental authority.

Also, we above defined the concept of a [System of Record](#) as an authoritative source. Think about the various kinds of data that might be needed in the case of disputes or legal matters:

- Employee records
- Sales records (purchase orders and invoices)
- Contracts and other agreements
- Key correspondence with customers (e.g. emails directing a stock broker to "buy")

These can be take the form of:

- paper documents in a file cabinet
- documents scanned into a document management system
- records in a database

In all cases, if they are "official" - if they represent the organization's best and most true understanding of important facts - they can be called "records."

This use of the word "records" is distinct from the idea of a "record" in a database. Official records are of particular interest to the company's legal staff, regulators, and auditors. Records management is a professional practice, represented by the Association of Records Management Administrators (www.arma.org).

Records management will remain important in digitally transforming enterprises, as lawyers, regulators, and auditors are not going away. One of the critical operational aspects of records management is the concept of the **retention schedule**.

It is not in a corporation's interest to maintain all data related to all things in perpetuity. Obviously, there is a cost to doing this. However, as storage costs continue to decrease, other reasons become more important. For example, data maintained by the company can be used against it in a lawsuit. For this reason, companies establish records management policies such as:

- Human Resources data is to be deleted 7 years after the employee leaves the company
- Point of Sale data is to be deleted 3 years after the transaction
- Real estate records are to be deleted 10 years after the property is sold or otherwise disposed of

This is not necessarily encouraging illegal behavior. Lawsuits can be frivolous, and can "go fishing" through a company's data if a court orders it. A strict retention schedule, that can be shown to be adhered to, can be an important protection in the legal domain.

IMPORTANT

If you or your company are involved in legal issues relating to the above, seek a lawyer. This discussion is not intended as legal advice.

We will return to records management in the discussion below on e-discovery and cyberlaw.

Records management drives us to consider questions such as "who owns that data" and "who takes care of it." This leads us to the concept of data governance.

Data Governance

This book views data governance as based in the fundamental principles of governance from [Chapter 10](#):

- Governance is **distinct** from management
- Governance represents a control and **feedback** mechanism for the digital pipeline
- Governance is particularly concerned with the **external environment** (markets, brands, channels, regulators, adversaries)

By applying these principles we can keep the topic of "data governance" to a reasonable scope. As [above](#) let's focus on the data aspects of:

- risk management, including security
- compliance
- policy
- assurance

Information related risks

The biggest risk with information is unauthorized access, discussed previously as a **security** concern. Actual destruction, while possible, is less of a concern given the nature of information (it can be copied and is intangible). Other risks include regulatory and civil penalties for mis-handling, and operational risks (e.g. from bad **data quality**.)

There are a wide variety of specific **threats** to data, leading to risk, for example:

- Data theft (e.g. by targeted exploit)
- Data leakage (i.e. unauthorized disclosure by insiders)
- Data loss (e.g. by disaster & backup failure)

The standard risk and security approaches suggested in Chapter 10 are appropriate to all of these. There are particular technical solutions such as data leakage analysis that may figure into a controls strategy.

A valuable contribution of information management is a better understanding of the risks

represented by data. We've discussed simple info sensitivity [models](#) (for example Public, Internal, Confidential, Restricted). However, a comprehensive information classification model must accommodate:

- basic sensitivity (e.g. Confidential)
- ownership/stewardship (e.g. SVP HR, HR/IS director)
- regulatory aspects (e.g. SOX or HIPAA)
- records management (e.g. "Human Resources," "Broker/Client Communications," "Patient History")

Beyond sensitivity, the regulatory aspects drive both regulatory and legal risks. For example, transmitting human resources data related to German citizens off of German soil is illegal, by German law. But if German HR data is not clearly understood for what it is, it may be transmitted illegally.

Other countries have different regulations, but privacy is a key theme through many of them. The U.S. HIPAA regulations are stringent in the area of U.S. medical data.

To thoroughly manage for such risks, data stores should be tagged with the applicable regulations and the records type.

The broad topic of individuals' expectations for how data relating to them is stored and secured is called *data privacy*. It drives regulations, lawsuits, standards, and is a frequent topic of news coverage (e.g. when a mass data breach occurs.)

Bad data quality also presents risks as mentioned above. In fact, [\[DeLuccia2008\]](#) sees data quality as a kind of [control](#) (in the sense of risk mitigation).

E-Discovery and Cyberlaw

Information systems and the data within them can be the subject of litigation, both civil and criminal. Criminal investigation may ensue after a security breach. Civil and regulatory actions may result from (for example) inappropriate behavior by the organization, such as failing to honor a contract.

In some cases, records are placed under a "legal hold." This means that (whether physical or digital) the records must be preserved. The United States Federal Rules of Civil Procedure [\[NationalCourt2016\]](#) covers the discovery of information stored in computing systems. Successfully identifying the data in scope for the hold requires disciplined approaches to records management and data classification, as described above.

IMPORTANT

Again, if you or your company are involved in legal issues relating to the above, seek a lawyer. This discussion is not intended as legal advice.

Analytics

Instructor's note:

This is a brief discussion, because many aspects of big data, analytics, and data science are specific forms of product development. This book for example has avoided detailed discussion of programming languages or current vendor products. Similarly, the intent here is to explore the background and management implications of big data, analytics, and data science, not to discuss specific techniques or products in any detail.

Analytics in context

...analytical competitors wring every last drop of value from business processes and key decisions.

— Tom Davenport and Jeanne Harris, Competing on Analytics

One important aspect of digital product development is data analytics. Analysis of organizational records has always been a part of any concern large enough to have formal records management. The word for this originally was simply "reporting." A set of files or ledgers would be provided to one or more clerks, who would manually review them and extract the needed figures.



Figure 14. Reporting and analytics, old style [24: Image credit

<https://www.flickr.com/photos/seattlemunicipalarchives/3347281230>, commercial use permitted]

We have touched on reporting previously, in our discussion of **metrics**. This is a more detailed examination.

The compilation of data from physical sources, and its analysis for the purposes of organizational strategy, was distinct from the day to day creation and use of the data. The clerk who attended to

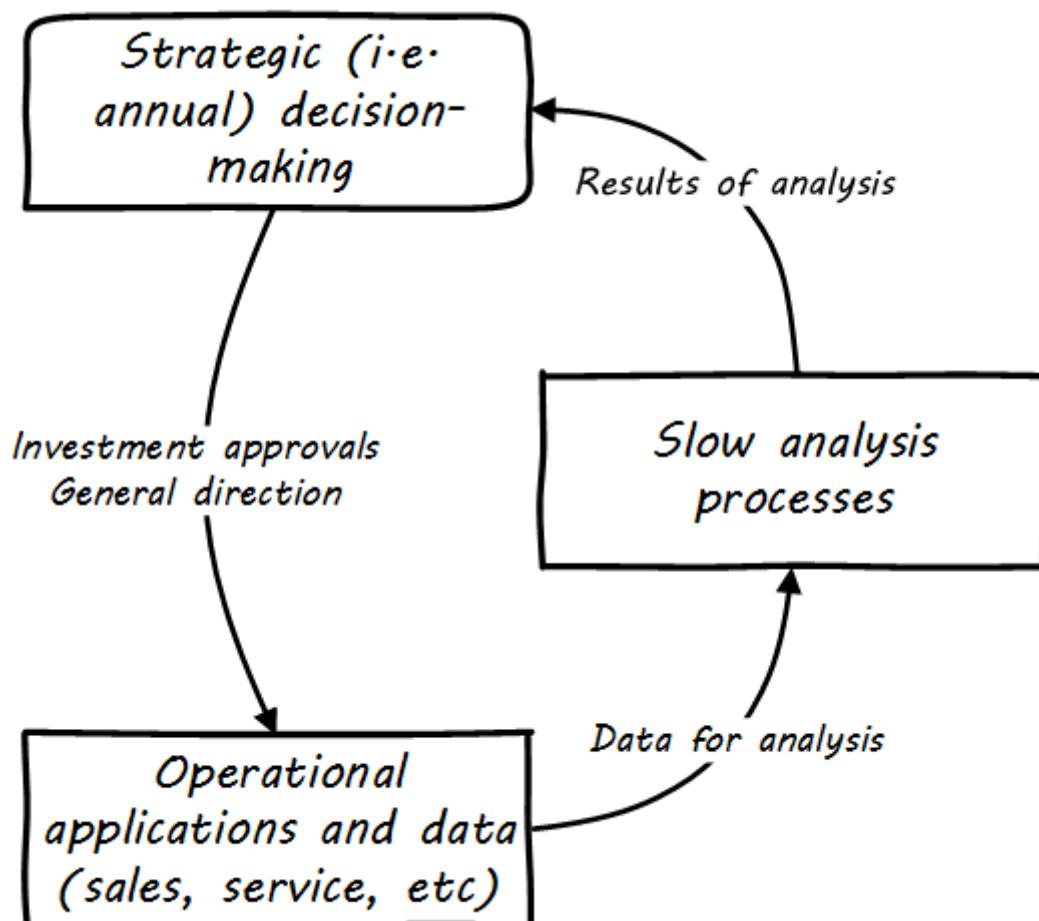
the customer and updated their account records, had a different role than the clerk who added up the figures across dozens or hundreds of accounts for the annual corporate report.

What do we mean by the words "analysis" or "analytics" in this older context? Just compiling totals and averages was expensive and time-consuming. Cross-tabulating data (e.g. to understand sales by region) was even more so.

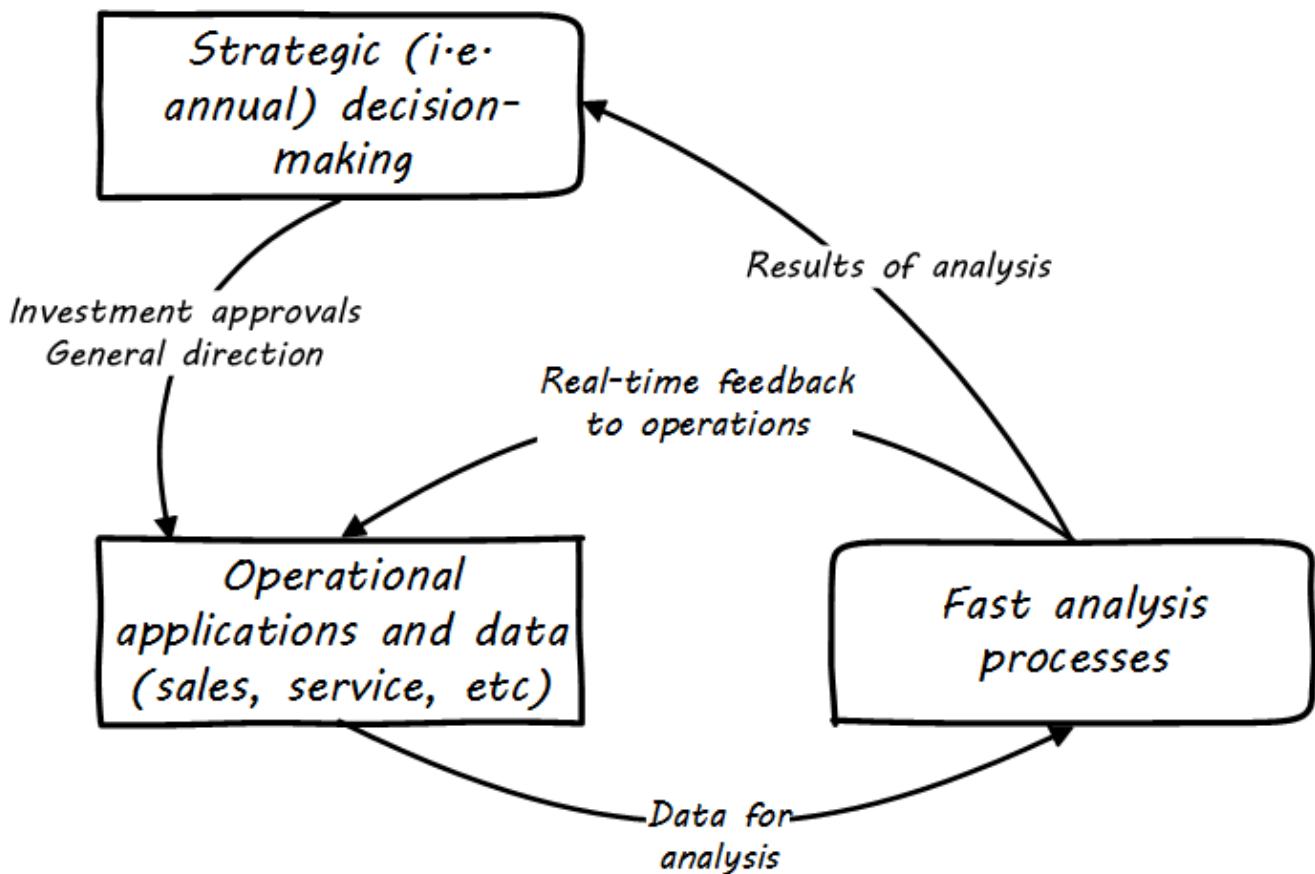
As information became more and more automated, the field of "Decision Support" (and its academic partner "Decision Sciences") emerged. The power of extensively computerized information, that could support more and more ambitious forms of analysis, gave rise to the concept and practice of "Data Warehousing" [Inmon1992].

A robust profession and set of practices emerged around data warehousing and analytics. And as infrastructure became more powerful and storage less expensive, the idea of full-lifecycle or closed-loop analytics originated.

When analyzing data is costly and slow, the data analysis can only affect large, long-cycle decisions. It is not a form of fast feedback. The annual report may drive next year's product portfolio investment decisions, but it cannot drive the day to day behavior of sales, marketing, and customer service:



However, as analysis becomes faster and faster, it can inform operational decisions.



And, for certain applications (such as an online traffic management application on your smartphone), analytics is such a fundamental part of the application that it becomes operational. Such pervasive use of analytics is one of the hallmarks of digital transformation.

Data warehousing and business intelligence

A Data Warehouse (DW) is a combination of two primary components. The first is an integrated decision support database. The second is the related software programs used to collect, cleanse, transform, and store data from a variety of operational and external sources...Data warehousing is a technology solution supporting Business Intelligence (BI).

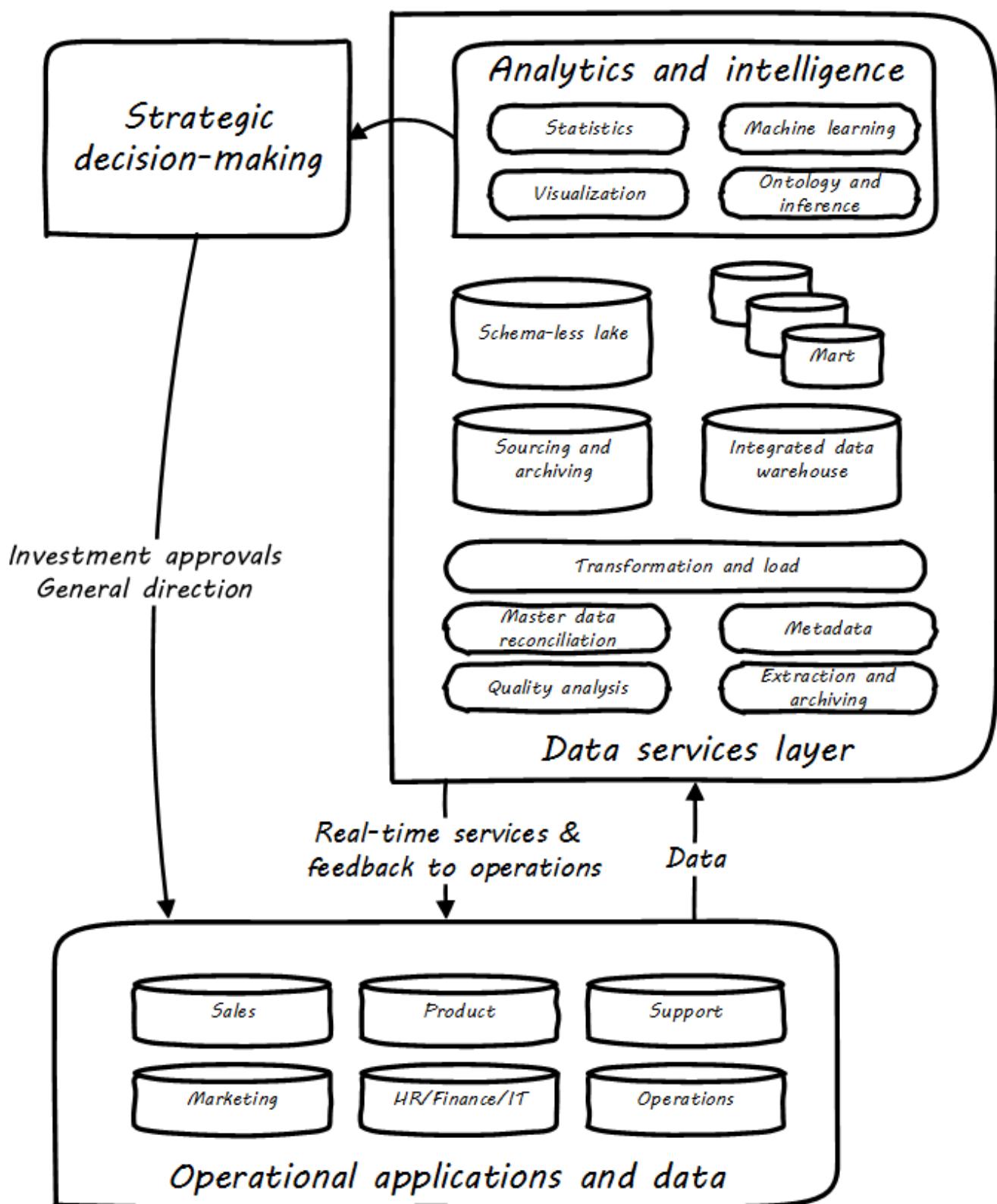
— Data Management Body of Knowledge

The reason to build a data warehouse is to have the ability to make better decisions faster based on information using current and historic data.

— Paul Westerman, Data Warehousing: Using the Wal-Mart Model

The vision of an integrated data warehouse for decision support is compelling and has provided enough value to support an industry sector of specialized hardware, software, training, and consulting. It can be seen as a common architectural pattern, in which disparate data is aggregated and consolidated for purposes of analysis, reporting, and for feedback into strategy, tactics, and operational concerns.

Here is an illustration of an data warehousing/business intelligence (DW/BI) implementation pattern:



The above diagram expands on the above [contextual diagrams](#), showing the major business areas (Sales etc) as data sources. (In a large organization this might be dozens or hundreds of source systems.) These systems feed a "data services layer" that both aggregates data for analytics, as well as providing direct services such as data cleansing and master data management.

It's important to understand that in terms of this book's emphasis on product-centric development that **the data services layer itself is an internal product**. Some might call it more of a [component than a feature](#), but it is intended in any case as a general-purpose platform that can support a wide variety of use cases.

"Factoring out" data services in this way may or may not be optimal for any given organization, depending on maturity, business objectives, and a variety of other concerns. However, at scale the skills and practices do become specialized, and so it's anticipated we'll continue to see implementation strategies similar to the above figure.

Notice also that the data services layer is not solely for analytics; it also supports direct operational services.

Here are discussions of the diagram's various elements:

Operational applications

These are the source systems that provide the data and require data services.

Quality analysis

This is the capability to analyze data for consistency, integrity, and conformity with expectations, and to track associated metrics over time. (See [data quality](#).)

Extraction and archiving

As data storage has become less expensive, maintaining a historical record of data extracts in original format is seen more often in data warehousing. (This may use a schema-less data lake for implementation.)

Master data reconciliation

When master data exists in diverse locations (e.g. in multiple [Systems of Record](#)) the ability to reconcile and define the true or "golden" master may be required. This is useful directly to operational systems, as an online service (e.g. postal service address verification), and is also important when populating the data warehouse or mart. Master data includes reference data, and in the data warehousing environment may be the basis for "dimensions," a technical term for the ways data can be categorized for analytic purposes (e.g. retail categorizes sales by time, region, and product line). Maintaining a history of dimensions is a challenging topic; search on the "slowly changing dimension" problem for further information.

Metadata

Commonly understood as "data about data," we have [previously](#) encountered the concept of metadata and will further discuss it in the next chapter section.

Transformation and load

Converting data to a consistent and normalized form has been the basis of enterprise data warehousing since it was first conceived. (We will discuss the [schema-less](#) data lake approach in the next chapter section.) A broad market segment of "Extract, Transform, Load" (ETL) tooling exists to

support this need.

Sourcing and archiving

This represents the physical data store required for the extraction and archiving capability. Again, it may be implemented as a schema-less data lake, or as a traditional relational structure.

Integrated data warehouse

The integrated or enterprise data warehouse is the classic, normalized, often massive, historical data store envisioned originally by Bill Inmon [\[Inmon1992\]](#). While the development effort in creating fully normalized data warehouses has limited them, they nevertheless are important, valuable, and frequently encountered in larger organizations.

Schema-less lake

A newer form of data aggregation is seen in the [schema-less](#) lake. As discussed in the next chapter section, schema-less approaches accept data in native formats and defer the hard question of normalizing the data to the reporting and analysis stage.

Mart(s)

The integrated data warehouse is intended to provide a consistent and universal platform across the enterprise. The data mart on the other hand is usually seen as specific to a particular organization or problem.

Statistics

Statistical analysis of the aggregated and cleansed data is a common use case, often performed using commercial software or the R programming language.

Machine learning

Machine learning is broadly defined as "a field of study that gives computers the ability to learn without being explicitly programmed." [Arthur Samuel as quoted in Simon, waiting on book]. Machine learning allows computers to develop and improve algorithmic models for making predictions or decisions. Spam filters that "learn" are a good example.

Visualization

Representing complex information effectively so that humans can understand it and derive value is itself a challenging topic. Many graphical forms have been developed to communicate various aspects of data. See for example the open source visualization library [D3.js](#)

Ontology and inference

This includes text mining and analytics, and also the ability to infer meaning from unstructured data sets. More in the next chapter section discussion on [schema-less](#).

Agile methods meet DW/BI

Data infrastructure, like any complex systems development effort, is most effective and least risky

when undertaken iteratively and incrementally. An organization's analysis needs will change unpredictably over time and so a fast feedback loop of testing and learning is essential.

The enterprise data warehouse can support a wide variety of analysis objectives flexibly. Its challenge has always been the lead time required to develop the data structures and ETL logic. This will be discussed further in the next chapter section.

Agile information management

Software versus data

Services and Applications can have their own Gravity, but Data is the most massive and dense, therefore it has the most gravity. Data if large enough can be virtually impossible to move. [\[McCrory2010\]](#)

— Dan McCrory

Enterprise information management (including data management) has had a contentious relationship to Agile methods. There are inherent differences in perspective between those who focus on implementing software, versus those who are concerned with data and information, especially at scale.

The tendency among software engineers is to focus on the conceptual aspects of software, without concern for the physical issues of computing. (Logical versus physical is arguably the fundamental distinction at the heart of the [development vs operations divide](#).)

As we have previously discussed, [refactoring](#) is one of the critical practices of Agile development, keeping systems flexible and current when used appropriately. The simple act of turning one overly-large software class or module into two or three more cohesive elements is performed every day by developers around the world, as they strive for clean, well engineered code.

DevOps and Agile techniques can be applied to databases; in fact there are important technical books such as Refactoring Databases by Ambler and Sadalage [\[Ambler2006\]](#). With smaller systems, there is little reason to avoid ongoing refactoring of data along with code.

Infrastructure as Code techniques apply. Database artifacts (e.g. SQL scripts, export/import scripts, etc.) must be under version control and should leverage continuous integration techniques. Test-driven development can apply equally well to database-related development.

But when data reaches a certain scale, its concerns start to become priorities. The bandwidth of FedEx is still greater than that of the Internet ([\[Munroe2013\]](#)). That is to say, it is more effective and efficient, past a certain scale, to physically move data by moving the hard drives around, than to copy the data. The reasons are well understood and trace back to fundamental laws of physics and information first understood by Claude Shannon [\[Shannon1949\]](#). The concept of "data gravity" (quote above) seems consistent with Shannon's work.

Notice that these physical limitations apply not just to simple movement of data, but also to complex [refactoring](#). It's one thing to refactor code—even the SQL defining a table. Breaking an existing, overly-large database table and its contents into several more specialized tables is a

different problem, if data is large. Data, in the sense understood by digital and IT professionals, is **persistent**. It exists as physical indications of state in the physical world: whether knotted ropes, clay tablets, or electromagnetic state.

If you are transforming 3 billion rows in a table, each row of data has to be processed. The data might take hours or days to be restructured, and what of the business needs in the meantime? These kinds of situations often have messy and risky solutions, that cannot easily be "rolled back":

1. A copy might be made for the restructuring, leaving the original table in place.
2. When the large restructuring operation (perhaps taking hours or days) is completed, and new code is released, a careful conversion exercise must identify the records that changed while the large restructuring occurred.
3. These records must then go through the conversion process AGAIN and be updated in the new data structure. They MUST replace the older data that was initially converted in step 1.

All in all, this is an error-prone process, requiring careful auditing and cross-checking to mitigate the risk of information loss. It can and should be automated to the maximum degree possible.

Modern web-scale architectures are often built to accommodate rolling upgrades in a more efficient manner (see [\[Limoncelli2014\]](#) Chapter 11, "Upgrading Live Services"). Perhaps the data can be transformed on a "if changed, then transform" basis, or via other techniques. Nevertheless, schema changes can still be problematic. Some system outage may be unavoidable, especially in systems with strong transactional needs for complete integrity. (See earlier discussion of [CAP theorem](#).)

Because of these issues, there will always be some contention between the software and information management perspectives. Software developers, especially those schooled in Agile practices, tend to rely on heuristics such as "do the simplest thing that could possibly work." Such a heuristic might lead to a preference for a simpler data structure, that will not hold up past a certain scale.

An information professional, when faced with the problem of restructuring the now-massive contents of the data structure, might well say "Why did you build it that way? Couldn't you have thought a little more about this?"

In fact, data management capabilities often have sought to intervene with developers, sometimes placing procedural requirements upon the development teams when database services were required. This approach is seen in [waterfall](#) development and when database teams are organized [functionally](#).

We saw the classic model [earlier](#) in this chapter: in turn, define

1. Conceptual data model
2. Logical data model
3. Physical schema

as a sequential process.

But organizations pressed for time often go straight to defining physical schemas. And indeed, if the

[Cost of Delay](#) is steep, this behavior will **not** change.

The only reason to invest in richer understanding of information (e.g. conceptual and logical modeling), or more robust and proven data structures, is if the benefits outweigh the costs. Data and records management justifies itself when:

- Systems are easier to adapt, because they are well understood and the data structures are flexible
- Occurrence, costs, and risks of data refactoring are reduced
- Systems are easier to use, because the meaning of their information is documented for the end consumer (reducing support costs and operational risks)
- Data redundancy is lessened, saving storage and management attention
- Data and records-related risks (security, regulatory, liability) are mitigated through better data management.

Again, back to our [emergence model](#). By the time you are an enterprise, faced with the full range of Governance, Risk, Security, and Compliance concerns, you likely need at least **some** of the benefits promised by Enterprise Information Management. But data management, like any [functional](#) domain, can become an end in itself, losing sight of the reasons for its existence.

Next generation practices in information management

Cross-functional teams

The value of cross-functional teams was discussed at length in [chapter 7](#). This applies to including data specialists as team members. This practice alone can reduce many data management issues.

Domain-driven design

As you try to model a larger domain, it gets progressively harder to build a single unified model [\[Fowler2014\]](#).

— Martin Fowler

The relational database, with its fast performance and integrated schemas allowing data to be joined with great flexibility, has fundamentally defined the worldview of data managers for decades now.

This model arguably reached its peak in highly-scaled mainframe systems, where all corporate data might be available for instantaneous query and operational use.

However, when data is shared for many purposes, it becomes very difficult to change. The analysis process starts to lengthen, and [Cost of Delay](#) increases for new or enhanced systems.

This started to become a real problem right around the time that cheaper distributed systems became available in the market. Traditional data managers of large scale mainframe database systems continued to have a perspective that "everything can fit in my database." But the demand for new digital product outstripped their capacity to analyze and incorporate the new data.

The information management landscape became fragmented. One response was the enterprise conceptual data model. The idea was to create a sort of "master schema" for the enterprise, that would define all the major concepts in an unambiguous way.

However, attempting to establish such a model can run into difficulties getting agreement on definitions. (See the [ontology problem](#) above.) Seeking such agreement again can impose cost of delay, if gaining agreement is required for the system. And if gaining agreement is optional, then why is agreement being sought? The risk is that the data architect becomes "ivory tower."

NOTE

In fact, there are theoretical concerns at the heart of philosophy with attempting to formulate universal ontologies. They are beyond the scope of this text but if you are interested, start by researching *semiotics* and *postmodernism*. Such concerns may seem academic, but we see their consequences in the practical difficulty of creating universal data models.

A pragmatic response to these difficulties is represented in the Martin Fowler quote above. Fowler recommends the practice of domain-driven design, which accepts the fact that "Different groups of people will use subtly different vocabularies in different parts of a large organization" [\[Fowler2014\]](#) and quotes Eric Evans that "total unification of the domain model for a large system will not be feasible or cost-effective" [\[Evans2004\]](#).

Instead, there are various techniques for relating these contexts, beyond the scope of this book. (See [\[Evans2004\]](#).) Some will argue for the use of microservices, but data always wants to be recombined, so microservices have limitations as a solution for the problems of information management.

And, before you completely adopt a domain-driven design approach, be certain you understand the consequences for data governance and records management. Human resources records are human resources records and must be handled appropriately. Regulators and courts will not accept "domain driven design" as a defense for non-compliance.

Generic structures and inferred schemas

Schema development - the creation of detailed logical and physical data and/or object models - is time consuming and requires certain skills. Sometimes, application developers try to use highly generic structures in the database. Relational databases and their administrators prefer distinct tables for Customer, Invoice, and Product, with specifically identified attributes such as Invoice Date. Periodically, developers might call up the database administrator and have a conversation like this (only slightly exaggerated):

"I need some tables."

"OK, what are their descriptions?"

"Just give me 20 or so tables with 50 columns each. Call them Table1 through Table20 and Column1 through Column50. Make the columns 5000-character strings, that way they can hold anything."

"Umm... You need to model the data. The tables and columns have to have names we can

understand."

"Why? I'll have all that in the code."

These conversations usually would result in an unsatisfied developer and a DBA further convinced that developers just didn't understand data.

A relational database, for example, will not perform well at scale using such an approach. Also, there is nothing preventing the developer from mixing data in the tables, using the same columns to store different things.

This might not be a problem for smaller organizations, but in organizations with compliance requirements, knowing with confidence what data is stored where is not optional.

This does not mean that the developer was completely off track. New approaches to data warehousing use generic schemas similar to what the developer was requesting. Speed of indexing and proper records management, can be solved in a variety of ways.

Recently, the concept of the "data lake" has gained traction.

Some data has always been a challenge to adapt into traditional, rigid, structured relational databases. Modern "web-scale" companies such as Google have pioneered new, less structured data management tools and techniques.

The data lake integrates data from a large variety of sources, but does not seek to integrate them into one master structure (also known as a schema) when they are imported. Instead, the data lake requires the analysts to specify a structure when the data is extracted for analysis. This is known as "schema-on-read," in contrast to the traditional model of "schema on write."

Data lakes, and the platforms that support them (such as Hadoop) were originally created high volume web data such as generated by Google. There was no way that traditional relational databases could scale to these needs, and the data was not transactional – it was harvested and in general never updated afterwards.

This is an increasingly important kind of workload for digital organizations. As the Internet of Things takes shape, and digital devices are embedded throughout daily experiences, high-volume, adaptable datastores (such as data lakes) will continue to spread.

Because log formats change, and the collaboration data is semi-structured, analytics will likely be better served with a "schema on read" approach. However, this means that the operational analysis is significant development. Simplifying the load logic only defers the complexity. The data lake analyst must have a thorough understanding of the various event formats and other data brought into the lake, in order to write the operational analysis query.

"Schema on read" still may be a more efficient approach however. Extensive schema development done up front may be invalidated by actual data use, and heavyweight approaches are not as compatible with fast feedback. (Data services are also a form of product development and therefore fast [feedback](#) on their use is beneficial; the problem again is one of data gravity. Fast feedback works in software because code is orders of magnitude easier to change.)

Schema inference at the most general shades into *ontology mining*. In ontology mining, data

(usually text-heavy) is analyzed by algorithms to derive the data model. If one reads a textbook about the retail business, one might easily infer that there are concepts such as "store," "customer," "warehouse," and "supplier." Information technology has reached a point where such analysis itself can be automated, to a degree. Certain analytics systems have the ability to display an inferred table structure derived from unstructured or semi-structured data. This is an active area of research, development, and product innovation.

cite

The challenge is that data still needs to be tagged and identified; **regulatory concerns do not go away** just because one is using a NoSQL database.

For further information and the current state of industry practice on these questions, see the professional associations at the end of this chapter.

Append-only to the rescue?

Another technique that is changing the data management landscape is the concept of append-only. Traditional databases **change** values; for example, if you change "1004 Oak Av." to "2010 Elm St." in an address field, the old value is (in general) **gone**, unless you have specifically engineered the system to preserve it.

A common approach is the idea of "audited" or "effective-dated" fields, which have existed for decades. In an effective-dated approach, the "change" to the address actually looks like this in the database:

Street address	From	To
1004 Oak Av.	12/1/1995	9/1/2016
2010 Elm St.	9/2/2016	Present

Determining the correct address requires a query on the To date field. (This is only an example, there are many ways of solving the problem.)

In this approach, data accumulates and is not deleted. (Capacity problems can of course result.) Append-only takes the idea of effective dating and applies it across the entire database. No values are ever changed, they are only superseded by further appends.

This is a powerful technique, especially as storage costs go down. It can be combined with the data lake to create systems of great flexibility. But there are no silver bullets. Suppose that a distributed system has sacrificed consistency for availability and partition-tolerance (see [CAP theorem](#)). In that case, the system may wind up with data such as:

Street address	From	To
1004 Oak Av.	12/1/1995	9/1/2016
2010 Elm St.	9/2/2016	Present
574 Maple St.	9/2/2016	Present

This is now a [data quality](#) issue, requiring after-the-fact exception analysis and remediation.

Append-only complements architectural and programming language trends towards *immutability*.

Test data

...when teams have adequate test data to run automated tests, and can create that data on demand, they see better IT performance

— Puppet Labs/DevOps Research and Assessment, 2016 State of DevOps Report

A non-obvious and non-trivial problem at the intersection of Enterprise Information Management and DevOps is test data management.

What is test data management?

Suppose you are a developer working on a data-intensive system, one that (for example) handles millions of customer or supply chain records.

Your code needs to support a wide variety of data inputs and outputs. At first, you just entered a few test names and addresses, like "Mickey Mouse" or "Bugs Bunny, 123 Carrot Way, Albuquerque, New Mexico 10001."

But this nonsensical data quickly was shown to not work. For example, if you are testing integration with an address-scrubbing service, you will get an error with an address in New Mexico that shows a ZIP code of 10001. (Actually, the nonsensical data is useful in testing that particular error scenario. But that is only one of many error scenarios.)

Based on hearing anecdotal concerns, the authors of the *2016 State of DevOps* report examined test data management practices and found that they correlated positively with "better IT performance, lower change failure rates, and lower levels of deployment pain and rework" [Forsgren2016], p. 29. In particular, the report suggests that test data be minimized and created from a blank slate wherever possible.

Taking data from production systems as a basis for testing is also frequently done. However, such data must be sanitized — sensitive information such as social security number must be removed. This can be done automatically, but then such automation must itself be developed and maintained, and the extensive production data set may (in effect) be driving a large amount of non-value-add testing.

In general, test data management techniques will vary greatly by application and problem domain. The primary recommendation here is to invest in solving the problem, understanding that up-front investments in automation will pay off. The high-performing product team will have to solve the "how" of doing it appropriately for their particular situation.

Non-invasive data governance (Seiner?)

Information management topics

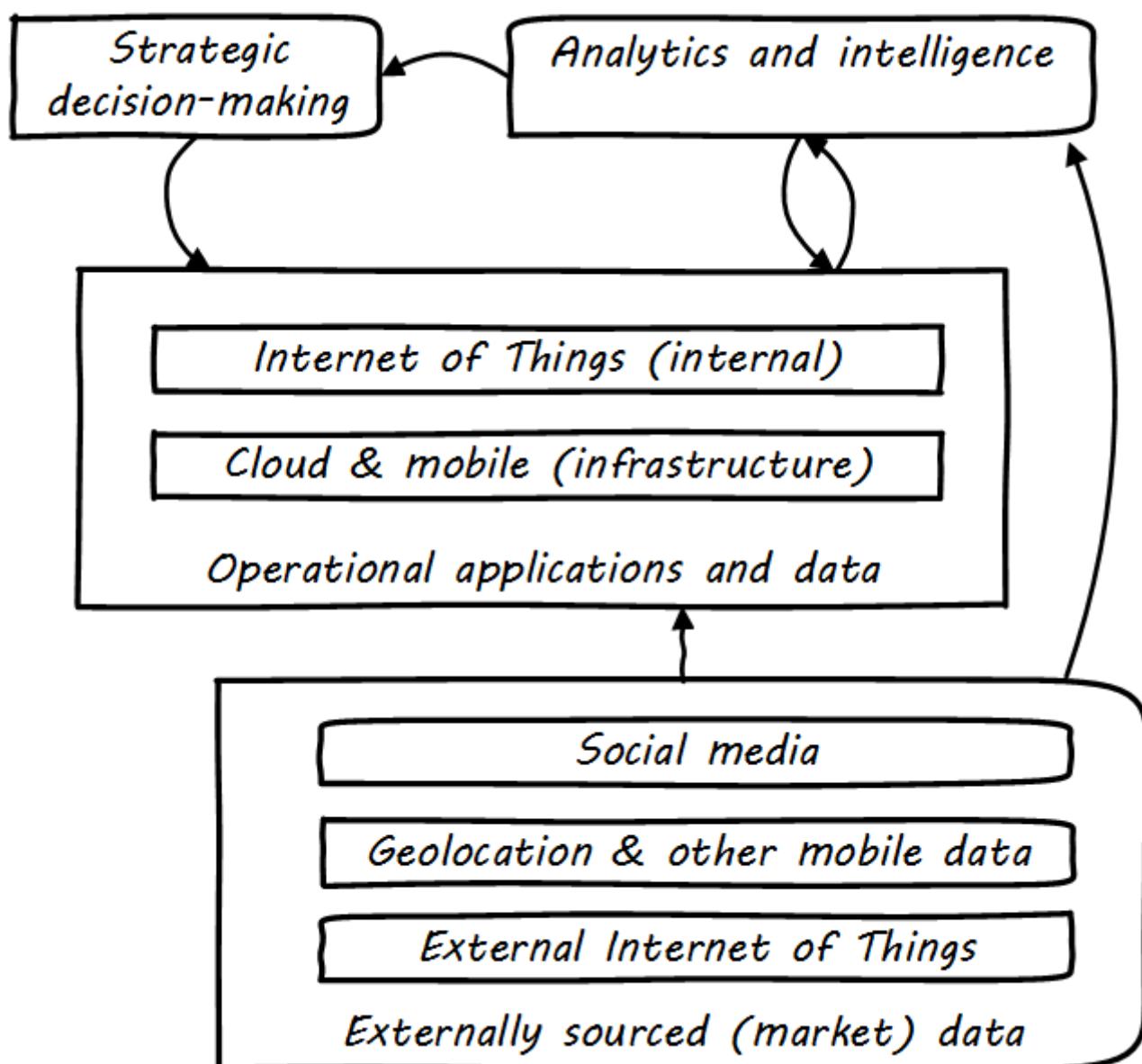
Social, Mobile, Analytics, and Cloud

Discussions of digital transformation often reference the algorithm SMAC:

- Social
- Mobile
- Analytics
- Cloud

Others would add the Internet of Things.

These are not equivalent terms; in fact, they have certain relationships to each other:



Social media is generally external to an organization and manifests as a form of **commercial data**, that provide essential insights into how a company's products are performing and being received.

Mobile (or mobility) has two distinct aspects: mobility as an engagement platform (e.g. for deployment of "apps" as one product form), versus the **commercial data** available from mobile carriers, notably geolocation data.

Internet of Things can be either an internal or external data source, often extremely high volume and velocity, requiring analysis services for sense-making and value extraction.

Big data

The term "Big Data" in general refers to data that exceeds the traditional data management and data warehousing approaches we have discussed above. As proposed by analyst Doug Laney in 2001 [[Laney2001](#)], its most well known definition identifies 3 dimensions of scale:

- Volume
- Variety
- Velocity

High volume data is seen for example in the search history logs of search engines such as Google.

High variety data encompasses rich media and unstructured data, such as social media interactions.

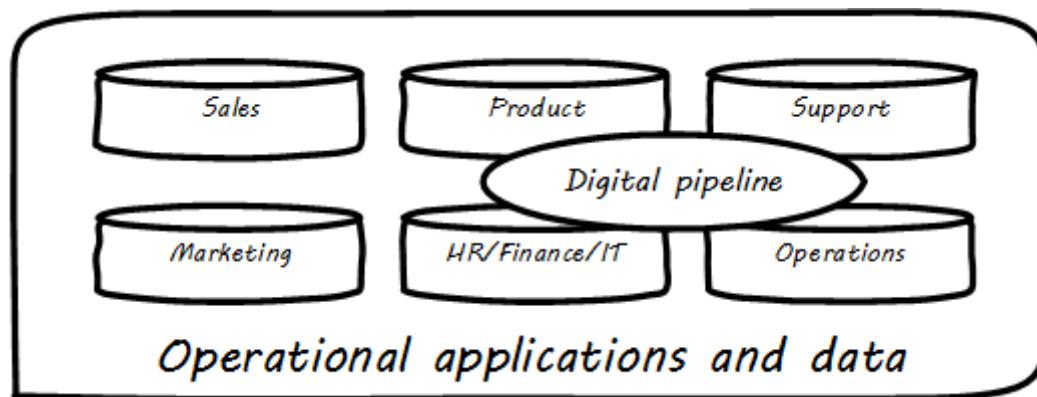
High velocity data includes telemetry from Internet of Things devices or other sources capable of emitting large volumes of data very quickly.

All of these dimensions require increasingly specialized techniques as they scale, and the data management product ecosystem has continued to diversify as a result.

Managing the information of digital delivery

In terms of our previous data warehousing architecture, the digital pipeline can be seen as related to four areas:

- Product management
- IT
- Support
- Operations



Assume the primary product of the organization is an information-centric digital service, based ultimately on data. How do you manage data? How do you manage anything? In part, through collecting data about it. Wait - "data about data"? There's a word for that: **metadata**. We'll take some time examining it, and its broader relationships to the digital delivery pipeline.

We've talked about **metadata** previously and you should review that section before continuing. The association of business definitions with particular data structures, is one form of metadata.

Data governance, records management, and ongoing support for digital consumers all require some layer of interpretation and context to enrich the raw data resource.

Consider the following list:

Server	Database
SRV001	DB0023
SRV001	DB0045
SRV002	DB0067

Not very useful, is it? Compare it to:

Server	Database	Product	Regulatory
SRV001	DB0023	Online reviews	Customer privacy
SRV001	DB0045	Employee records management	HIPAA, PII
SRV002	DB0067	Online sales	PCI, PII

We could also include definitions of the tables and columns held in each of those databases. But what system would contain such data? There have been a couple primary answers over the years: metadata repositories and configuration management databases.

In terms of this book's general definition of **metadata** as non-runtime information related to digital assets, both the metadata repository and the CMDB contain metadata. But they are not the only systems in which data related to digital services is seen. Other systems include monitoring systems, portfolio management systems, risk and policy management systems, and many more. All of these systems themselves can be aggregated into a data warehousing/BI closed-loop infrastructure.

This means that the digital organization, including organizations transforming from older IT-centric models, can apply Big Data, machine learning, text analytics, and the rest of the techniques and practices covered in this chapter.

NOTE

We will examine a full "business of IT/digital delivery pipeline" architecture in the next chapter.

Additional topics

===== Data science
===== Semantic Web & ontologies
===== ERP systems (special section??)
===== Business rules management
===== Knowledge mgmt
===== Information lifecycle (POSMAD)
McGilvary (based on English) pp 23-24 - perhaps into 4.11.03.

Schema.org & phonetics transcription from CACM 02-2106

Conclusion

Information management is far more than creating databases. At scale, it is a challenging topic invoking fundamental questions of human understanding and meaning. It drives specialization and non-trivial governance issues, and has inherent contrasts with the perspectives of software development.

The exabytes of stored data accumulated by the human race represent an asset of incalculable value. Organizations that invest in and cultivate the ability to understand and effectively use this data to create information and actionable knowledge may prosper and serve their stakeholders well.

Discussion questions

- Identify a situation where you can put a price tag on having or not having information.
- As a group, each of you list at least 5 organizations who you know to store significant data about you. Compare and discuss.
- Have you ever had your data poorly handled, e.g. either corrupted or exposed? Do you have any privacy concerns? Compare experiences and discuss.
- Have conflicting definitions of some word ever caused you problems or led to interesting experiences? Discuss.

Research & practice

- Develop a simple conceptual data model of some aspect of your experience: hobbies, interests, extracurricular activities, etc. How would you build a small database for a social club, for example?
- Implement your concept as a simple database, e.g. in Microsoft Access or equivalent.
- Work through [An Introduction to R](#).
- Build a minimum viable demonstration of some [D3.js](#) visualization.
- Research "slowly changing dimensions" and summarize the alternatives for solving the problem.

Further reading

Books

- Scott Ambler, [Refactoring Databases](#)
- Alex Berson, [Master Data Management and Data Governance](#)
- [The DAMA Guide To The Data Management Body Of Knowledge](#)
- Danette McGilvray, [Executing Data Quality Projects: Ten Steps to Quality Data and Trusted Information](#)
- Gregor Hohpe, [Enterprise Integration Patterns](#)
- William Kent, [Data and Reality](#)
- Sydney Padua, [The Thrilling Adventures of Lovelace and Babbage: The \(Mostly\) True Story of the First Computer](#)

Articles/Reports

- [Agile methodologies in EDM projects](#)
- [BoundedContext](#) (alternative to the full enterprise conceptual model or ontology)
- [Falsehoods Programmers Believe about Names](#)
- [Applied Infonomics: Why and How to Value Your Information as an Asset](#)
- [DHS Report No. 2006-03, The Use of Commercial Data](#)

Videos

Professional

- [Data Management Association](#)
- [Association of Records Management Administrators](#)
- [The Data Warehousing Institute](#)

Chapter 12: Architecture and Portfolio

Collaborative

Status: Chapter 12 is in first draft as of 25 October 2016. Comments appreciated.

- [Collaborator instructions](#)
- Raise an [issue](#) to comment
- [Github source files](#)

Please remember that the intended audience of this one or two semester survey text is incoming students at a Masters' level in Information Technology, Information Systems, and Software Engineering programs. The most useful feedback comes from the perspective of, "what do you think today's students need to learn, given the last 20 years' developments with Agile, Lean IT, DevOps, web- and cloud-scale IT, digital transformation, and related themes?"

Introduction to Chapter 12

Regardless of whether you started off as a digital startup, or are an older corporation now digitally transforming, your world is complex and getting more so.

Decisions that might have been made quickly and casually by a product or project team become more difficult, due to increasing challenges both internal and external. Decisions made years ago come back to haunt current strategies with a vengeance. With scale, management needs some way of making sense across digital operations of mind-numbing complexity. Should we invest in 15 new internally-developed microservices, or sunset 12 existing ones and implement a commercial package developed by people we trust? When we have 1500 applications or services in our portfolio, how do we know that the proposed number 1501 is not redundant or outright conflicting with existing services?

Architecture provides tools to manage such problems, but doing so is difficult and controversial. Is the architecture merely drawings of [HiPPoS](#)? How can we maintain an experimental and hypothesis-driven approach in the midst of all the complexity? How do we know that our understanding of the digital operation is current and up to date? How much should we spend on keeping it so? What happens when management decides to set direction using these same abstractions, and architects find themselves now enforcing what had originally started as mere explanation and sense-making?

[Vendor relationships](#) become a two-edged sword, providing increased value with access to higher levels of vendor resources, but at the cost of increased lock-in. Pure open-source strategies inevitably give way to monetized relationships, as the risk of not having support becomes unacceptable.

Portfolio management in terms of information technology means looking at long-lived IT investments in terms of their overall benefit, cost, and risk to the organization. This can and should be done regardless of whether the IT investment is external or internal facing. Products, services, and applications are the most useful portfolio constructs, although assets, technology products, and even projects also figure into longer-horizon value management. Enterprise architecture benefits

from a tight alignment with IT portfolio management, as well; it's not clear that a firm boundary between them could be drawn.

This chapter is in a way summative: it reflects all of the concerns discussed through the previous 11 chapters. Every chapter represents topics of interest to the architect. Now, we discuss a language and way of thinking to bring these concerns all together.

IMPORTANT

As with other chapters in the later part of this book, we are going to introduce this topic "on its own terms." We will then add additional context and critique in subsequent sections.

Chapter 12 outline

- Introduction
- Why architecture?
 - Defining Enterprise Architecture
 - Architecture organization
 - The value of EA
- Architecture practices
 - Architecture and governance
 - Architecture as a management program
 - Modeling and visualization
 - Repositories and knowledge management
 - The "rationalization" quest
- Architecture domains
 - Architecture perspectives
 - Architecture layers
 - Other forms of architecture
- Agile and architecture
 - The hubris of architecture
 - The hubris of Agile
 - Towards reconciliation
- Portfolio management
 - Application value analysis
 - Application rationalization
- Architecture standards
 - Business architecture
 - TOGAF

- Archimate
- Industry verticals
- Governmental frameworks

Chapter 12 learning objectives

- Identify the value of architecture
- Describe basic architectural practices
- Define and distinguish between various forms of architecture
- Characterize the friction between Agile and architecture, and techniques for resolving it

Why architecture?



Figure 15. St. Vitus Cathedral, Prague [25: Image credit

https://www.flickr.com/photos/aigle_dore/6365091333, downloaded 2016-10-25, image copyrights Moyan Brenn, commercial use permitted]

Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints. Architecture must include engineering considerations, so that the design will be economical and feasible; but the emphasis in architecture is upon the needs of the user, whereas in engineering the emphasis is upon the needs of the fabricator. [\[\[Buchholz1962\]\]](#), first published use of the term "architecture" with respect to computing system organization.]

— Fred Brooks, Planning a Computer System: Project Stretch

The word "architecture" is usually associated with physical construction: buildings, landscapes, civil infrastructure, and so forth. It was appropriated by systems engineers at IBM around 1960 to describe the problems of designing complex information processing hardware and software. This leads to some confusion, and occasional questions from "real" architects as to why IT people are calling themselves "architects." Perhaps a different choice of word would have been advisable.

In our journey to date, we have covered:

- Business [motivations](#) and [context](#),
- [Infrastructure](#)
- [Applications](#),
- [Products](#) and/or services,
- [Organization](#) and [process](#), and
- [Data and information](#).

We have also covered the practices by which ideas and intentions are established and translated by investment into actions, including:

- Investment of time and resources towards digital objectives (e.g. through [project management](#)),
- Acquisition of technologies and external services (i.e. [sourcing](#)),
- [Hiring of employees](#),
- [Governance](#) of digital organizations for [risk](#), [security](#), and other purposes.

As we have progressed in our journey, and scaled our company up, all these areas have continued to evolve. Specialization emerges. You have people with deep experience in Cloud architectures, and people with deep experience in e-records management and compliance. You don't have too many who are deep in both.

Your product portfolio (internal and external) is now in the hundreds or thousands. Some were built with the latest technology, and others run on older technologies now perceived to be dead ends. But investing in rewriting or replatforming them would not provide as much value as other uses of the funds, so you have to manage the risk of the older technology.

Investment decisions become harder. You're far beyond the days when you had a one-product

focus. You have multiple interacting products and multiple interacting teams, and the relationship between the teams and products is not one to one. Understanding the business case for the investment gets harder; when you have a thousand services over multiple business units, how do you know if someone is proposing a redundant new one?

And then there are the big headaches. A major commercial product version is going off support, and it's the perfect time to think about a rewrite or replatform (say into the Cloud). But the moving pieces and interdependencies are formidably complex, and if you get the analysis wrong the business impact will be severe. You acquire another firm, with a lot of overlapping activities, and start to see the need for "Business Architecture" to clarify your understanding of business processes, organizations, and capabilities. Or a major outage hits the business hard, and all of a sudden the organizational priority (from the Board on down) is "fix this so it never happens again." Everything else is to go "on hold." Except, of course, it can't.

In response to these and a thousand other complexities of digital management as organizations scale up, a general purpose coordination capability emerges sometimes called Enterprise Architecture. In this chapter section, we will discuss its definition, organizational dynamics, and value proposition.

Defining Enterprise Architecture

The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

— ISO/IEC IEEE 42010

The enterprise architecture is the organizing logic for business processes and IT infrastructure, reflecting the integration and standardization requirements of the company's operating model. The enterprise architecture provides a long-term view of a company's processes, systems, and technologies so that individual projects can build capabilities—not just fulfill immediate needs.

— Jeanne Weill and Peter Ross, *Enterprise Architecture as Strategy*

Enterprise architecture (EA) is the representation of the structure and behavior of an enterprise's IT landscape in relation to its business environment. It reflects the current and future use of IT in the enterprise and provides a roadmap to reach a future state. [Bente2012] p 35

— Bente et al, *Collaborative Enterprise Architecture*

"Architecture" as a term by itself is something you've encountered since your earliest days as a startup. Perhaps you used it to describe the choice of technologies you used for your products. Or

the most important components in your application. Or the common services (e.g. authentication) you developed to support multiple products. The architecture concept is therefore not something new or foreign. But what does it mean to say we have an "enterprise" architecture? **Enterprise architecture is nothing but the unification of this book's topics into a common, formalized, scalable framework for understanding.** It means we are "doing architecture" comprehensively, considering the enterprise itself as a system to be architected. It also may mean we have a program for sustaining the work of those doing architecture in the technical, application, solution, data, process, or business domains.

In terms of our emergence model, enterprise architecture assumes multi-product, "team of teams" problems. As an overall domain of practice, enterprise architecture encompasses a variety of specialist domains (some of which we've already encountered) as we'll discuss in the next chapter section. Some of those domains **do** make sense at smaller, single-product contexts (e.g., software architecture.)

There are numerous definitions of *enterprise architecture*. The examples above are typical. It can be defined as:

- an organizational unit
- an organizational capability
- a formalized program
- a professional discipline or set of practices
- a process or process group; a ongoing activity or activities
- a large-scale artifact (i.e. an integrated model consisting of catalogs, diagrams, and matrices) maintained on an ongoing basis for communication and planning
- an integrated and standardized language for reasoning about complexity

In general, definitions of enterprise architecture characterize it as a coordination and problem solving discipline, suited to large scale problems at the intersection of digital technology and human organization. An important function of architecture is supporting a shared mental model of the complex organization; we were first introduced to the importance of **shared mental models** in Chapter 5 and this need has only increased as the organization became more complex. Enterprise Architecture provides the tools and techniques for sustaining shared mental models of complexity at scale.



Figure 16. The terrain [26: Image credit <https://www.flickr.com/photos/endogamia/3979040177>, downloaded 2016-10-16, commercial use permitted]



Figure 17. The mapmaker [27: Image credit <https://www.flickr.com/photos/perspective/8474999166>, downloaded 2016-10-16, commercial use permitted]



Figure 18. The map [28: Image credit https://www.flickr.com/photos/martye_green/2354576085, downloaded 2016-10-16, commercial use permitted]

Confused? Consider the activity of map-making. In map-making you have:

- The actual terrain
- The capability of map making: surveying, drawing, etc.
- The process of surveying and mapping it
- The resulting map as a document

And once the map is made, you might use it for a wide variety of purposes, and you also might find that once you start to use the map, you wish it had more information. Similarly, in enterprise architecture, it's important to remember that there are different concerns:

- Operational reality
- The capability of representing it for planning and analysis ("being" an architect or an architecture organization; having the skills and tools)
- The process of representing and analyzing the operational reality ("doing" architecture)

- The actual representation (the "architecture" as a "thing" - a model, an artifact, etc.)

And, like a map, once you have the architecture, you can use it for a wide variety of purposes, but also you may find it incomplete in various ways.

Architecture organization

There are three major themes we'll discuss in terms of the overall organizational positioning of enterprise architecture:

- The line versus staff concept and its origins
- Contrasting the concepts of "business model" versus "operating model"
- The other major organizational units of interest to enterprise architecture

Architecture as staff function

... the organization is weak at the top because there is no coordination of the exercise of powers provided for in the system. That coordination can be done only by a body organized for that purpose and having no other duties to perform; and in all the armies of the civilized world that duty is performed by a General Staff. They are called this because their duties are staff duties pertaining to the general conduct of affairs, and not merely to the work [of specific departments].

— Elihu Root, 1903 congressional testimony on US army re-organization (condensed)

We saw in [Chapter 10](#) how governance [emerges](#), as a [response](#) to scale and growth, and the concerns for [risk](#) and [assurance](#) in the face of increasing pressures of the external environment.

Architecture has a comparable emergence story, but for somewhat different reasons. Consider the quote from Elihu Root, above. Root was tasked with re-organizing the US Army at the turn of the 20th century. There had been some embarrassing failures of coordination and organization, and it was clear that something was missing in the US military: a centralized coordination function responsible for planning and logistics.

These lessons had also been learned by the Austrian, Prussian, and French militaries.



Figure 19. Franz Moritz Graf von Lacy [29: Image credit

https://en.wikipedia.org/wiki/Franz_Moritz_von_Lacy#/media/File:Count_Franz_Moritz_von_Lacy_oil_on_canvas_portrait_HGM.jpg, downloaded 2016-10-04, labeled as public domain by Wikipedia]

In previous ages, nobles would assemble and fight for their King, but their armies were poorly coordinated, and disputes over strategy would often arise. Each noble's army would have its own quartermaster, couriers, intelligence, supply chains, and the like. This was both inefficient, and ineffective: overall plans of battle often could not be made or executed, and military operations would be bungled.

As fighting wars became (unfortunately) a larger and larger scale endeavor, the need to centralize certain capabilities became more and more obvious. This culminated in the late 1700s and early 1800s with the creation of "general staffs" that were responsible for coordination of planning and execution across the increasingly complex military operations. The HQ "staff" and its shared services were deliberately distinguished from the warfighting "line." The Austrian field marshall Franz Moritz Graf von Lacy was one of the pioneers of this new style of organization, which was soon copied by the French army under Napoleon Bonaparte.

For example, the role of Napoleon's staff officer Pierre-Joseph Bourcet under Napoleon is described thus:

'On every occasion when an important decision had to be made Bourcet would write a memorandum in which he analyzed the situation and set forth in detail, with full explanations and reasons, the course which seemed to him best. In very many cases, his suggestions were adopted and were usually justified by success, and when they were rejected the results were seldom fortunate.' [Kiley2001]

Why do obscure French and Austrian military officers have to do with today's digital organizations? Because the line/staff distinction became the basis for large organizations of all kinds. As Christian Millotat (and many others) have noted, "[m]any elements that have become integral parts of managerial economics and organizing sciences can be traced back" to military staff systems ([Millotat1992], p. 7). These include:

- Collecting and combining knowledge so that decisions are as well informed as possible
- Supporting specialized roles and functions (e.g. legal experts, engineers)
- Operating supply chains and other services that function best when shared

Staff functions in the enterprise include planning, coordination, and operations; broadly speaking, and with key differences depending on the industry, the following are considered "staff":

- Financial management
- Human resources management
- Legal services
- Purchasing & vendor management (varies w/company and industry; for example in retail "merchandising" is a line function)
- Information technology (however, with digital transformation this is increasingly overlapping with R&D and driven directly by line management)
- Facilities management
- Strategic planning & forecasting

While the following are considered "line" (analogous to the warfighting units in the military):

- Sales
- Marketing
- Operations
- Research and development (varies w/company and industry)

Enterprise architecture has as a key part of its mission the task of collecting and combining knowledge to support decision-making. Therefore, an Enterprise Architecture organization can be seen as a form of staff organization. Most often it is seen as a specialized staff function within the larger staff function of IT management, and with the increased role of digital technology there are corresponding pressures to "move EA out of IT" as we'll discuss below.

The classic line/staff division is a powerful concept, pervasive throughout organizational theory. But it has important limitations:

- Staff organizations can "lose touch," become insular and self-serving, and indeed accumulate power in dangerous and unaccountable ways. For this reason, officers are rotated between line and staff positions in the US military.
- Staff "expertise" may matter less and less in **complex and chaotic** environments requiring experimental and adaptive approaches.
- If a **feedback** loop involves both line and staff organizations, it risks being delayed. The delay waiting for "headquarters approvals" has been a common theme in line/staff organizations.

In the history of line versus staff relations, we see tensions similar to those between Enterprise Architecture and advocates of Agile methods. The challenges, debates, and conflicts have only changed in their content... not their essential form.

But in many cases, centralizing staff expertise and the definition of acceptable practices for a domain is still essential. See the discussion of [matrix organizations](#) and even [feature versus component](#) teams.

Enterprise architecture and the operating model

The operating model is the necessary level of business process integration and standardization for delivering goods and services to customers. Different companies have different levels of process integration across their business units (i.e., the extent to which business units share data). Integration enables end-to-end processing and a single face to the customer, but it forces a common understanding of data across diverse business units.

— Weill and Ross, *Enterprise Architecture as Strategy*

In terms of overall positioning, Enterprise Architecture is often portrayed as mediating between strategy and portfolio management:

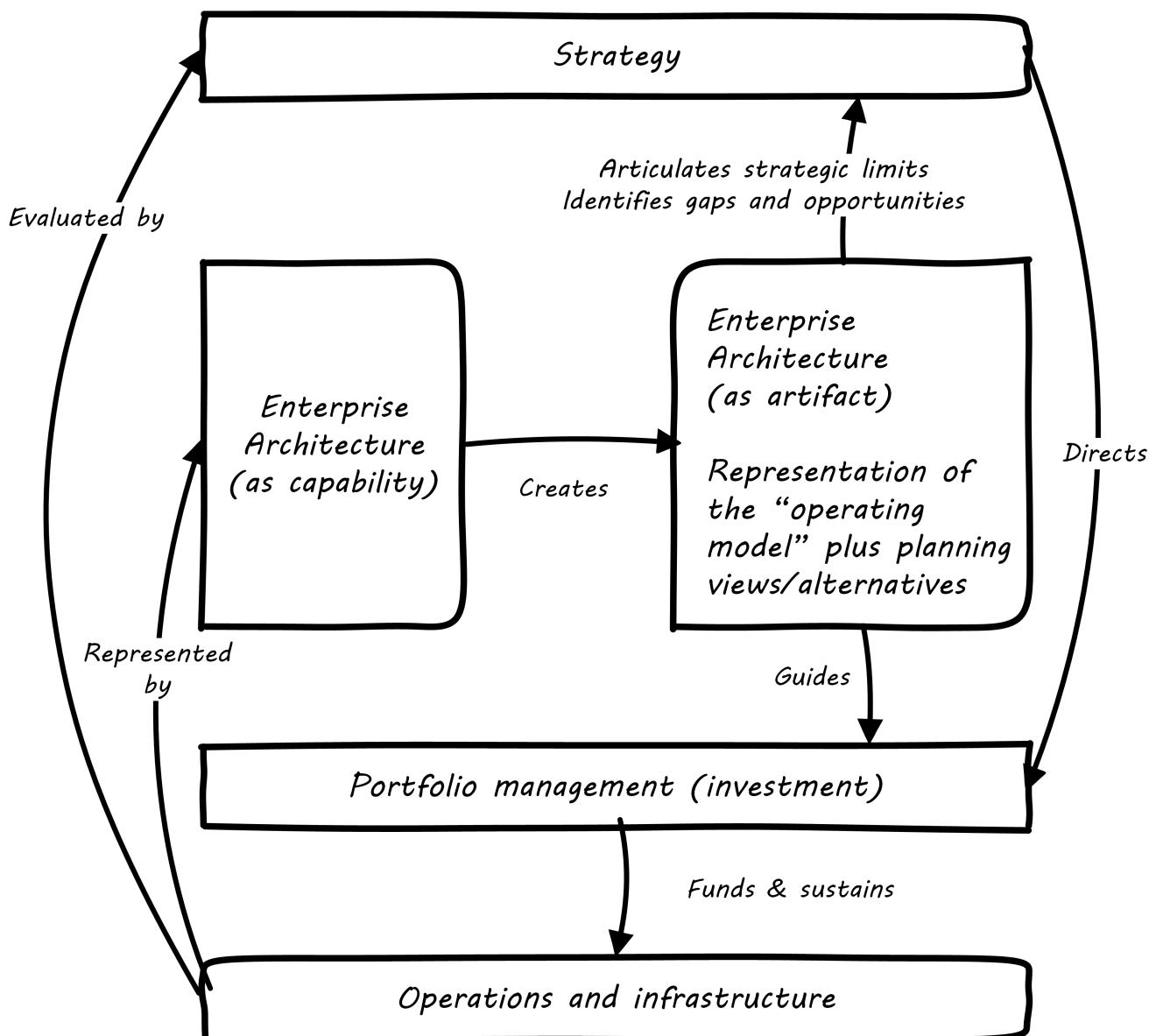


Figure 20. EA context, based on [Ross2006], fig 1-2, p.10.

Notice the distinction between Enterprise Architecture as a capability and as an artifact. The **practice** of Enterprise Architecture is not the same as the actual Enterprise Architecture. For the purposes of this textbook, we define enterprise architecture's concerns as essentially the enterprise operating model: process, data, organizational capabilities, and systems.

Author's note

While the concept of an "operating model" is somewhat loose, I believe there is a general consensus that it is distinct from a "business model." I also do not think that there is much value in distinguishing between the "enterprise architecture" (as an artifact) and the "enterprise operating model," especially at the higher levels of the enterprise architecture.

Scott Bernard (author of *An Introduction to Enterprise Architecture*) claims that: [without enterprise architecture] "...leadership will not have the ability to generate clear, consistent views of the overall enterprise on an ongoing basis, they won't be able to effectively compare business units, and the locus of power for planning and decision-making will be at the line-of-business, program, and/or system owner levels-with significant differences in how things are done and high potential for overlapping or duplicative functions and resources." [\[Bernard2012\]](#), introduction.

Now, truth be told, corporations compare business units all the time, on profit and loss and other enterprise metrics, **without** an architecture. If an enterprise is a holding company, by definition it is not seeking a common operating model. So, it is important to understand the role of enterprise architecture within the context of the enterprise strategy and business model, which remain distinct.

But, assuming that some shared vision, intelligence, economies of scale, and shared services are part of the business model, the concept of an operating model is a powerful tool for categorizing critical information, and identifying redundancies, overlaps, and potential synergies.

One of the most frequently used visualizations of enterprise architecture's concerns is the Zachman Framework:

	<i>What</i>	<i>How</i>	<i>Where</i>	<i>Who</i>	<i>When</i>	<i>Why</i>
<i>Contextual</i>	Glossary	Value chain	Markets	Partners Competitors	Calendars	Strategies
<i>Conceptual</i>	Conceptual data model	Business processes	Locations	Org chart	Cycles	Mission
<i>Logical</i>	Logical data model	System architecture	Network	RACI	Cadence	Initiatives
<i>Physical</i>	Physical schema	Software architecture	Nodes	Employees	Events	Objectives & key results
<i>Outcome</i>	Information	Code	Availability	Performance	Delivery	Execution

Figure 21. A variation on the Zachman framework

(loosely based on [\[Zachman1987\]](#) and succeeding work.)

We were exposed to the [data modeling](#) progression from conceptual to logical to physical data model in Chapter 11. The Zachman framework generalizes this progression to various views of importance to organizations, as shown in the columns:

- What
- How
- Where
- Who
- When
- Why

Overall, the Zachman framework represents the range of organizational operating model concerns well. Certainly, sustaining a large and complex organization requires attention to all its concerns. But what good does it do to simply document the contents of each cell? Such activity needs to have relevance for organizational planning and strategy; otherwise, it is just waste.

Peer organizations

It's reasonable to associate the [Business Model Canvas](#) with strategy, and the Zachman framework with the enterprise architecture (as an artifact):

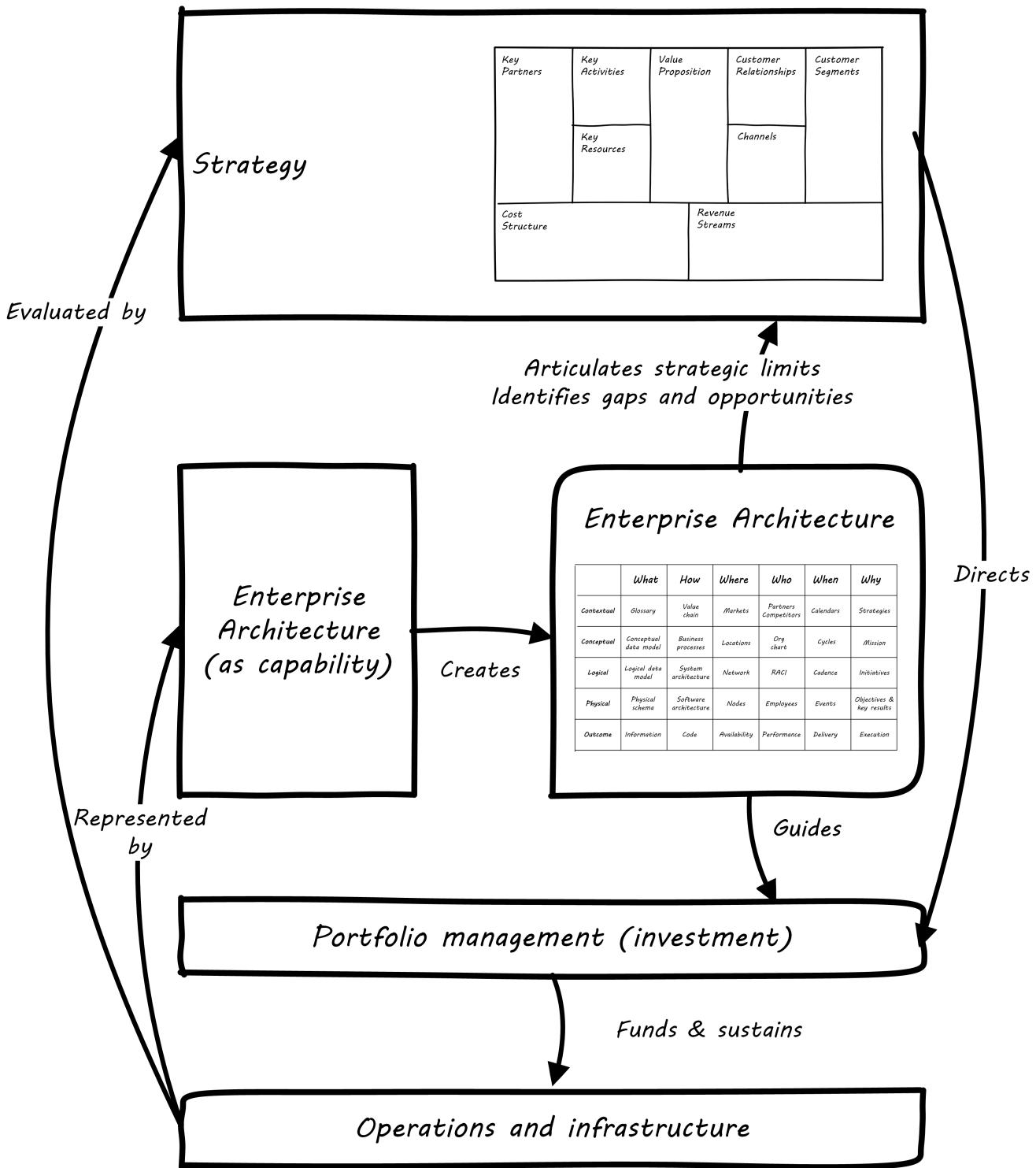


Figure 22. Business model versus operating model

This distinction helps us position the Enterprise Architecture group with respect to key partner organizations:

- Organizational strategy
- Portfolio and investment management
- Infrastructure and operations

Organizational strategy

The operating model needs to support the business model, and so therefore Enterprise Architecture

needs a close and ongoing relationship with organizational strategists, whether they are themselves line or staff.

Defining digital strategies is a challenging topic; we have touched on it in [Chapter 1](#), [Chapter 4](#), and Chapter 8. Further discussion at the enterprise level will be deferred to a future edition of this book.

Portfolio and investment management

Architecture needs to be tied to the organization's investment management process. This may be easier said than done, given the silos that exist. As Scott Bernard notes, "Enterprise architecture tends to be viewed as a hostile takeover by program managers and executives who have previously had a lot of independence in developing solutions for their own requirements" ([\[Bernard2012\]](#), case study scene 1).

Many organizations have a long legacy of project-driven development, in which the operational consequences of the project were too often given short shrift. The resulting [technical debt](#) can be crippling. Now that there is more of a move towards "you build it, you run it" the operability aspects of systems are (perhaps) improving. However, ongoing scrutiny and management are still needed at the investment front end, if the enterprise is to manage important objectives like vendor leverage, minimizing technical debt, reducing investment redundancy, controlling the security perimeter, and keeping skills acquisition economical (more on this below in section on Enterprise Architecture value).

[Portfolio management](#) is discussed in depth in a subsequent chapter section.

Infrastructure and operations

Finally, the EA group often has a close relationship with [infrastructure and operations](#) groups. This is because in organizations where operations is a shared service, the risks and inefficiencies of technical fragmentation are often most apparent to the operations team.

In organizations where operations is increasingly distributed to the application teams ("you build it, you run it") the above may be less true.

Other staff organizations that may develop close relationships with enterprise architecture include [vendor management and sourcing](#), [risk management](#), [compliance](#), and [security](#). Notice that some of these have strong [governance](#) connections (although we do not consider "governance" itself to be an organization, which is why it was not included in the discussion above.)

The value of EA

On the value side, EA is unique in its ability to promote enterprise-wide thinking about resource utilization...EA promotes the development of more efficient enterprise-wide common operating environments for business and technology, within which more capable and flexible business services and systems can be hosted. This in turn makes an enterprise more agile and able to respond to internal and external drivers of change, which promotes greater levels of competitiveness in the marketplace. [\[Bernard2012\]](#), chapter 3

— Scott Bernard

As a hygiene factor, benefits from EA can be valued in terms of reduction in management escalations, emergency occurrences, and year-on-year operational expenses. As a strategic foundation, EA facilitates the deployment of new capabilities. [\[Bente2012\]](#)

— Bente et al, Collaborative Enterprise Architecture

Enterprise architecture often struggles to demonstrate clear, quantifiable value to the organization. Architects are usually among the most experienced and therefore expensive staff in the organization. While it may seem to make historical and intuitive sense that architecture as a staff function is necessary, demonstrating this takes some thought and effort. Statements like "promoting enterprise-wide thinking" easily provoke skepticism. What are the benefits of so-called "enterprise-wide thinking"? And who receives them?

The following outcomes are often asserted for enterprise architecture:

- Shortening planning & decision-making (e.g. through curating information)
- Curating a shared enterprise language and mental model
- Increased speed of delivering new functionality
- Reduced and simplified portfolios
- Reducing duplication and re-work
- Reducing headcount (e.g. in processes)

The following model is a high level [impact mapping](#) representation.

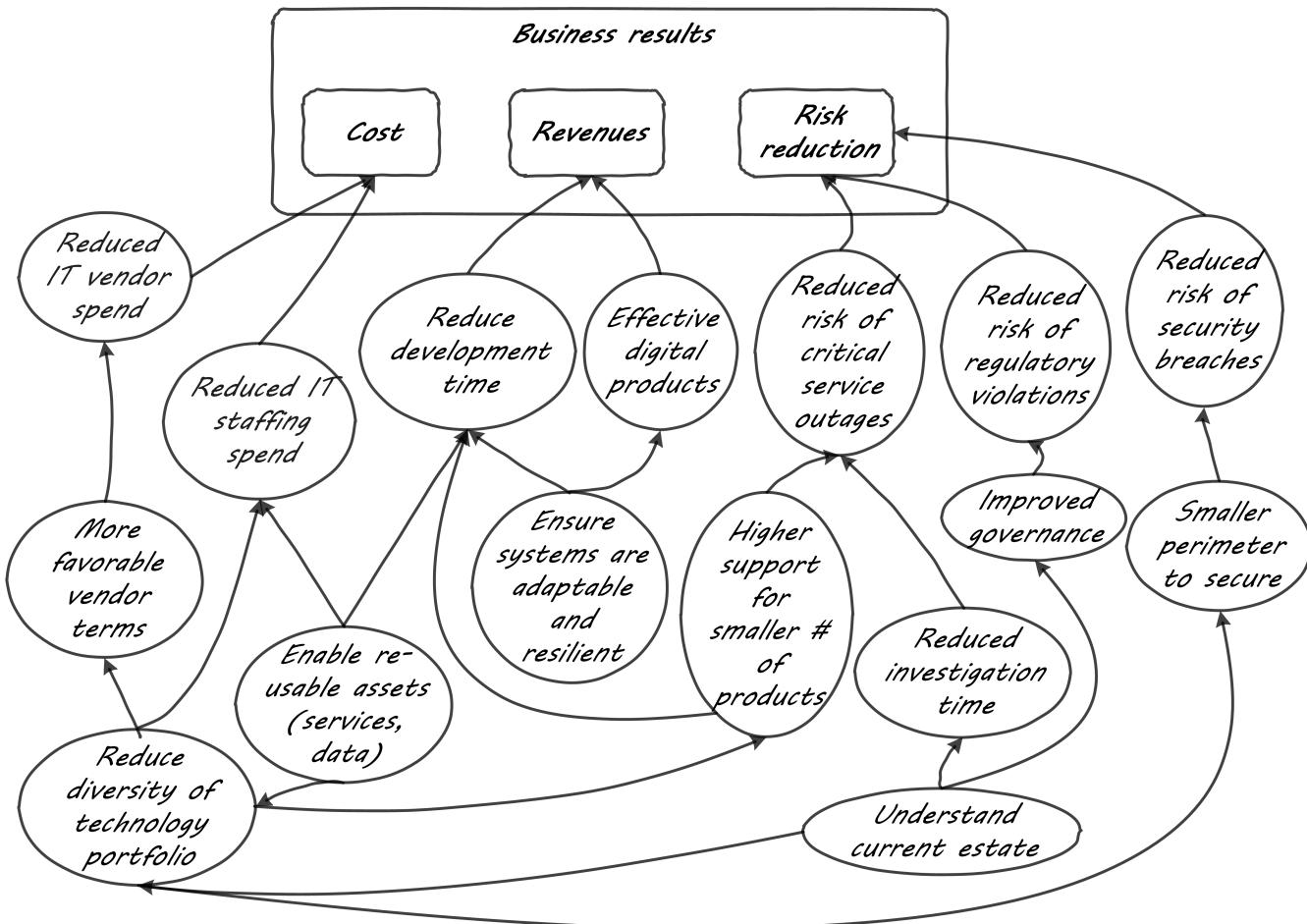


Figure 23. Architecture impacts on enterprise value

The above diagram suggests a number of specific, measurable outcomes from typical enterprise architecture goals. Without exploring every line of value:

- A reduced technology portfolio can and should result in improved [sourcing](#), improved support, improved security, reduced IT staffing spend, and potentially reduced development time. For example, vendors may offer more favorable terms when their products are preferred standards throughout an organization. A smaller product portfolio is easier to [secure](#).
- Better understanding of the current estate should reduce investigation times and outages, and reduce the risk of regulatory violations. For example, if regulators require evidence that employee medical [records](#) have not been removed from the country, architecture's curating of that information will expedite [compliance](#) response.
- Ensuring systems are adaptable (e.g. they have service interfaces) and resilient (they are designed for operability) should improve both [time to market](#) and ultimately effectiveness in customer acquisition and retention.

These are not intangible suggestions. We have previously studied the work and influence of [Don Reinertsen](#), who emphasizes the critical importance of an economic model.

IMPORTANT

Some might say that architecture's value is "intangible." If you are tempted to say this, you should read Douglas Hubbard's *How to Measure Anything: Finding the Value of "Intangibles" in Business* [[Hubbard2010](#)].

We close this chapter section by discussing two current value concepts and how architecture

contributes (or detracts) from them:

- Cost of Delay
- Technical Debt

Reducing Cost of Delay

We have covered the concept of [Cost of Delay](#) previously, at the team and product level. However, this powerful concept can also be applied at higher levels. Cost of Delay is not a concept familiar to most architects. It poses two important challenges:

- How can architecture help reduce Cost of Delay within the product portfolio?
- How can architecture not, itself, **introduce** un-economical Cost of Delay?

As we've discussed previously, the definition of Cost of Delay is intuitive. It is the opportunity cost of **not** having a given product or service available for use: the foregone revenues, the cost of the workarounds and inefficiencies. If the Architecture process becomes the critical path for a product or service's release (a common experience), then the Architecture process is responsible for that product's or services' Cost of Delay.

Cost of Delay can take various forms, some of the significant. For example: suppose there is a need to demonstrate a product to key clients at a trade show. This could be the company's best opportunity to develop business; the sales team estimates \$12 million in funnel opportunities based on previous experience that should result in at least \$2 million in sales in the year, with projections of another \$1 million in maintenance and renewals. However, if the product is not ready, these benefits will not materialize. If everything else is ready, but the Architecture process is delaying product readiness, then the Architecture process is incurring \$2 million in the cost of delay. This is bad. The Architecture process is clearly impacting significant business objectives and revenue.

But the question still needs to be asked, "what benefits do we receive from having an Architecture process?" We discussed such benefits above. Are these benefits adding up to \$2 million a year? No? Then your Architecture process does not make good economic sense.

On the other hand, what if the architects were kept out of the picture, and the product team chooses an untested technology, instead of re-using a well known and reliable approach already proven for that company? What if that decision were the cause of missing the trade show? What if it can be shown that re-usable components identified as architecture standards were increasing the speed of delivery, and reducing the Cost of Delay, because they are reducing the need for product teams to perform risky (and yet redundant) Research and Development activities?

Quantifying these benefits across a portfolio is difficult, but should be attempted. Cost of Delay can and should be calculated at the portfolio level, and this can provide "enterprise-level decision rules" that can help an organization understand the cost and value of operating model changes ([\[Reinertsen1997\]](#), pp. 35-38) including instituting processes (such as Change Management or Technology Lifecycle Management), or even the establishment of Enterprise Architecture itself.

Technical debt

Although immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation...

— Ward Cunningham, OOPSLA '92

Technical debt is a metaphor first introduced by Ward Cunningham [[Cunningham1992](#)] in the context of software, widely discussed in the industry. It can be applied more broadly at the portfolio level, and in that sense is sometimes discussed in enterprise architecture. Debt exists in the form of obsolete products and technologies; redundant capabilities and systems; interfaces tightly coupled where they should be loose and open; and many other forms.

Technical debt, like Cost of Delay, can and should be quantified. We'll discuss approaches to that in the chapter section on [portfolio management](#).

Scaling the enterprise mental model

"standard EA language and methodology is especially helpful in large, complex enterprises that are geographically dispersed, and which may have multiple social and work cultures that have promoted different ways of doing things." [[Bernard2012](#)], chap.3.

— Scott Bernard, Introduction to Enterprise Architecture

We've often referred to the concept of [common ground](#) through this book. Architecture supports common ground understanding at scale, by curating a shared mental model for the organization. In doing so, it enables the "right emergent behaviors" (as [Adrian Cockcroft suggests](#)). It also enables communication across diversity, and may improve staffing flexibility and mobility among teams.

Architecture practices

Before we get into a detailed discussion of architecture domains, let's talk in general about what architects do and some common practices and themes.

As we mentioned [previously](#), architecture itself as a term shows up in many ways, as role, artifact, program, and organization.

In this chapter section, we will look at:

- The relationship of architecture and governance
- Architecture as a management program
- The importance of visualization as a practice in architecture
- Architecture and the quest for "rationalization"

Author's corner

What do architects do, actually?

I was a Lead Architect (job class Systems Architect 6) for Wells Fargo for 6 years. This is a director-level position, and I was ultimately awarded the VP title, which in financial services tends to be honorary. (I never managed a large organization.) My assigned domain was the "business of IT," including systems management, service management, portfolio management, and the data, processes, and tools of architecture itself.

Across an annual IT spend of around \$5 billion, many of course were involved in all these questions, but I was a nexus and was often called on to lead workshops and discussions with senior executives. These folks were typically at the "2-down" level from the CIO, that is, executives reporting to one of the CIO's direct reports, themselves leading large teams of teams. They called me in when they were struggling with managing one of the larger IT shops on the planet, especially when it came to processes, data, and systems **internal to the IT** organization. (I always admit, I never learned that much about banking; there was plenty to do just coordinating that IT pipeline and the thousands of systems it was delivering.)

Comparing my activities during those 6 years with how industry standards (such as TOGAF) describe architecture is interesting. Building models and "architectures" was only a small part of my work. I found myself involved in, or playing the role of:

- IT strategic planning
- IT portfolio management
- "Zoning authority," e.g. defining what systems are "[of record](#)" for what data
- Program architect, e.g. Application Decommissioning and Portfolio Rationalization for the Wells Fargo/Wachovia merger
- Internal consultant ("first line of defense before calling Accenture")
- Technology standards governance
- RFP owner for selecting new technologies
- Data governance
- Vendor product analysis (often in liaison with analyst firms such as Gartner Group)
- Center of Excellence for data, process (BPM), and systems analysis and modeling
- Solutions design standards and patterns
- Continuous service improvement (called "critical systems review")

- Major incident root cause analysis
- Project governance
- IT ombudsman and process troubleshooter ("you architects say the process should work this way, but it doesn't - help me")
- Shuttle diplomat - driving agreement between deeply conflicted business units or individuals, where relationships had broken down.

If I had to characterize the important skills for this role, they would start with communication and facilitation. In terms of the principles discussed in this book, the architecture team was a mechanism for synchronizing across the organization. It definitely fit the classic definition and value proposition of a [staff organization](#).

Architecture and governance

Enterprise architecture has a clear relationship to [governance](#) as we discussed it in Chapter 10. It provides a framework for managing [long-lifecycle concerns](#) and various forms of enterprise [risk](#), especially as related to digital and IT systems.

Architecture is an important part of the governance equation. Architecture becomes the vehicle for technical standards that are essential risk controls; a risk management organization cannot achieve this alone.

EA therefore may have a role in defining [policies](#), especially at the mid-tier of the [policy hierarchy](#) - neither the highest enterprise principles, nor the most detailed technical standards, but rather policies and standards related to:

- Choice of certain enterprise products expected to be heavily leveraged (e.g. common database and middleware products)
- Design patterns for solving recurring requirements (e.g. user authentication, load balancing, etc)
- [System of Record](#) identification and enforcement

As discussed in [Chapter 10](#), there needs to be traceability from tactical standards to strategic codes and principles. The preference for a given database should not be a policy, but having a process that establishes such a preference **would** be—that is, a policy should exist saying (for example) "there shall be a Technology Lifecycle Management process with the following objectives and scope." Where appropriate, such policies might also be linked to specific risks as [controls](#) or [enablers](#).

As for all policies, it's important to have some sort of sunset mechanism for EA guidance. As Bente et al note, *Many EA-originated policies that appear obsolete today have not always been meaningless...A frequent example is the uncontrolled proliferation of newly hyped technologies by the IT crowd, and the EA group's rigid attempt to reinstitute order. Once the technology has matured, the EA rules often appear overly strict and suppress a flexible use of the appropriate technology.* [\[Bente2012\]](#) p19

The issue with the quote above is that the overall benefits of having (for example) a Technology

Lifecycle Management process are not usually quantified in terms of cost and risk avoidance. Without an overall governance mandate and value proposition, Enterprise Architecture activities may seesaw in response to the "issue of the moment." This is not a recipe for sustainable architecture; whose most important value proposition lies in the long term. Architecture, as a component of coherent governance, requires no less.

As we discussed in Chapter 10 governance emerges in part as a response to [external forces](#). Architecture often plays a consultative role when external forces become governance issues, for example:

- Data custody and [Systems of Record](#), and relationship to [records management](#)
- Vendor relationship strategies
- Security risks and controls

Governance is also concerned with efficiency, which also becomes a key architecture concern with associated practices:

- IT portfolio rationalization
- Business process optimization
- Shared services and APIs re-use
- Master and [reference data management](#)

Finally, does enterprise architecture promote effectiveness? Effectiveness is often seen as the primary responsibility of "the line" in line/staff paradigms. However, as the [impact](#) model suggests, establishing a foundation of re-usability and limiting technical choices can increase the speed with which new products and services are delivered.

Architecture as a management program

The above section discussed the relationship of architecture to governance. As we covered in chapter 10, [governance is not management](#). Here, we will cover the topic of architecture as a management program of activity, in part through examining an example large-scale architecture program.

Architecture as a program refers to a coordinated set of

- processes,
- job roles,
- standards and practices,
- artifacts,
- organizations, and
- cadenced and ad-hoc activities

intended to serve a key [coordination](#) role. The below diagram represents a large scale, coordinated architecture program in a large enterprise. Notice that this is not a single organization. The Architecture Program in this example spans a centralized Enterprise Architecture group as well as

teams of Line of Business architects.

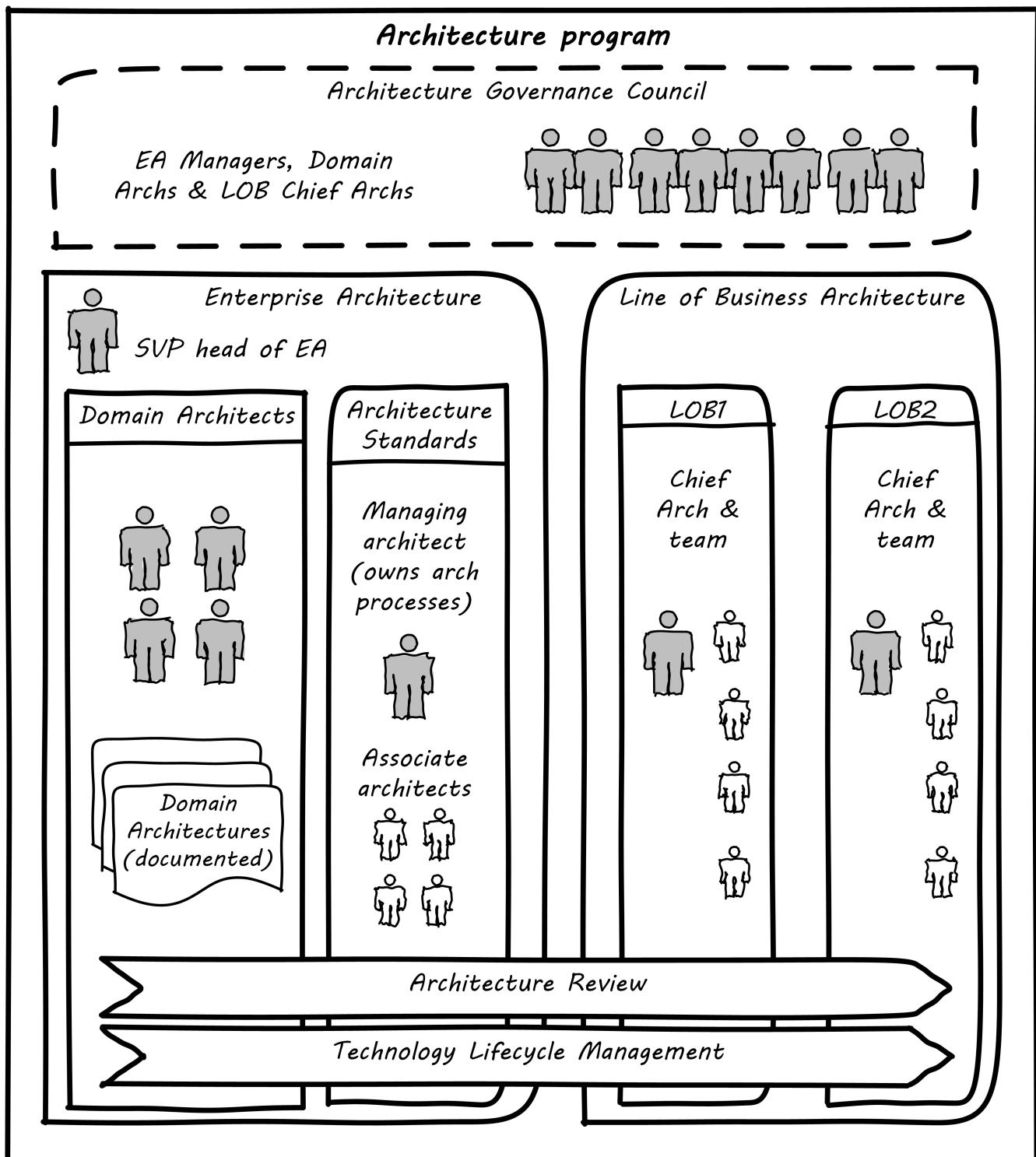


Figure 24. Large scale architecture program

The Enterprise Architecture organization might report to a CTO, the Chief of Staff for the Office of the CIO, or the head of Corporate Strategy and Planning. It is a centralized organization with a small staff of domain architects and an Architecture Standards organization that owns two key cross-functional architecture processes.

Lines of Business have dedicated IT organizations, and these organizations have Chief Architects with their own staffs. In terms of our discussion of [line/staff](#) organization, it is as if the line organization has its own staff function within it; another way to think about it is that the line/staff division is *fractal* (that is, it reproduces at different scales).

Within the central Enterprise Architecture organization, we have a number of director-level Domain Architects. These architects might focus on particular business problems (e.g. Supply Chain) or architectural domains (e.g. Data and Information, or Security).

It is the responsibility of the Domain Architects to create Domain Architectures, which are documents that lay out an overall point of view on a particular domain and often serve as standards. These architectures may be created according to a methodology such as TOGAF, with the support of a repository-based tool and language such as Archimate or various standards from the Object Management Group.

The domain architects also serve as a senior consulting pool and are assigned to significant programs and projects as needed.

The Architecture Standards organization is responsible for two organization-wide architecture processes:

- Architecture Review
- Technology Lifecycle Management

The **Architecture Review** process is part of the investment process, when initiatives are initially scoped and direction set. The process requires architects to review significant proposed investments in new systems for consistency with standards (e.g. the Domain Architectures and approved technologies). In terms of the previous section's **impact** model, this process is attempting to support many of the lines of value through controlling redundancy, and ensuring re-use and application of previously learned architectural lessons.

The **Technology Lifecycle Management** process is the means by which new vendor and open source products are approved as fit for purpose and/or preferred within the organization. In terms of the previous section's **impact** model, this process is tasked with reducing the portfolio of vendor products which reduces cost and risk as shown.

Both of these processes are enterprise-wide processes. They are owned, defined, and modified by the Architecture Standards organization, but projects and products across the enterprise follow these processes.

Finally, the **Architectural Governance Council** brings together the senior architects from the central EA organization and the LOB Chief Architects. It is a virtual organization operating on a quarterly **cadence**, responsible for setting direction and resolving the most difficult questions that may emerge from the architecture processes and domain architectures.

Overall, this may seem like a complex structure, but similar structures are in place in IT organizations with budgets of \$1bn or more. It would be questionable to see comparable structures in much smaller organizations. However, this structure is useful to examine; organizations of various sizes might choose to use different parts of it.

Modeling and visualization



Figure 25. Gudea with blueprint, ~2140 BCE [30: Image

https://www.flickr.com/photos/daryl_mitchell/16189447931, downloaded 2016-10-10, commercial use permitted]

The true measure of the value of a model is whether it actually influences behavior.

— Preston Smith and Don Reinertsen, Developing Products in Half the Time

We discussed the importance of visual management in Chapter 5. Making information visually available to help create **common ground**, is an important Lean practice (see [Andon](#)).

The word "architect," whether in a building or digital context, is often associated with visualizations: blueprints, sketches, specialized notations, and so forth. Drawings have been used to represent structures for likely as long as [writing has existed](#). The image at the beginning of this section is of Gudea, a Mesopotamian ruler known for building temples; on his lap is a blueprint.

Judging simply by its history, visualization is therefore an essential tool for humans dealing with large scale complexity (and erecting buildings has always been one of the more complex domains of human activity.) In digital and IT contexts, however, visualization has certain challenges and notable skeptics. Adrian Cockcroft, the former CTO of Netflix, stated: "Our architecture was changing faster than you can draw it... As a result, it wasn't useful to try to draw it" [[Bloomberg2014](#)].

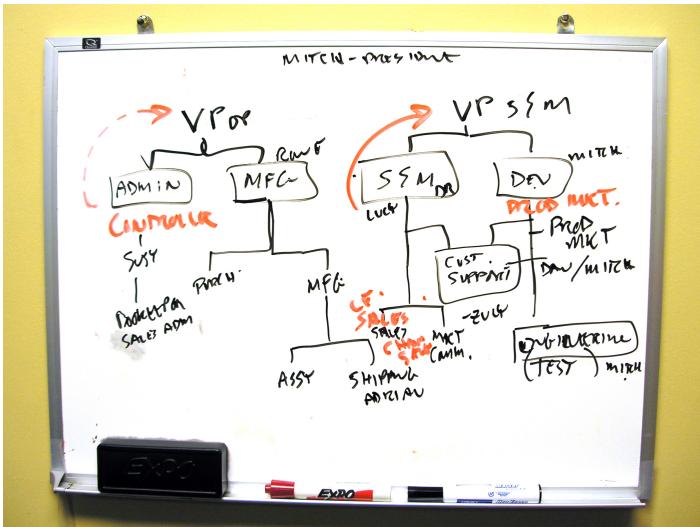


Figure 26. Whiteboard [31: Image <https://www.flickr.com/photos/simonov/15484240880>, downloaded 2016-10-10, commercial use permitted]

Even in construction and engineering trades that rely on blueprints as a source of truth, keeping them up to date requires considerable discipline and process. In faster-moving digital organizations, visual models are almost always out of date unless they have been specifically refreshed for a purpose, or unless there is a strong formal process in place (and the value of such a process may be difficult to establish). That doesn't mean that diagrams will go away. Co-located teams use whiteboards and dry-erase markers and will continue to use them. There are important cognitive and human factors reasons for this that will not go away. Because of these facts, it is useful to understand some of the fundamentals of how humans interpret visual data.

Human visual processing

Visual representations are effective because they tap into the capabilities of the powerful and highly parallel human visual system. We like receiving information in visual form and can process it very efficiently: around a quarter of our brains are devoted to vision, more than all our other senses combined [63]. In addition, diagrams can convey information more concisely [27] and precisely than ordinary language [8, 68]. Information represented visually is also more likely to be remembered due to the picture superiority effect [38, 70] ...Visual representations are also processed differently: according to dual channel theory [80], the human mind has separate systems for processing pictorial and verbal material. Visual representations are processed in parallel by the visual system, while textual representations are processed serially by the auditory system... [Moody2009].

— Dan Moody, The "Physics" of Notations

As the above quote shows, there are clear neurological reasons for diagramming as a communication form. To expand a bit more on the points Dan Moody is making:



Figure 27. Fast recognition means survival [32: Image

<https://www.flickr.com/photos/samiksha/2436037856>, downloaded 2016-10-10, commercial use permitted]

- Human vision uses parallel processing. This means that a given image or visual stimulus is processed by many neurons simultaneously. This is how we can quickly recognize and act on threats, such as a crouching tiger.
- A large percentage of our brain is devoted to visual processing (you will see figures quoted from 25% to 66% depending on whether it's "pure" visual tasks or vision-driven tasks involving other brain areas)
- The old saying "a picture is worth a thousand words" is consistent with the science. Diagrams can be both faster and more precise at conveying information; however, this has limits.
- Pictures can be more memorable than words.

Visualization in digital systems

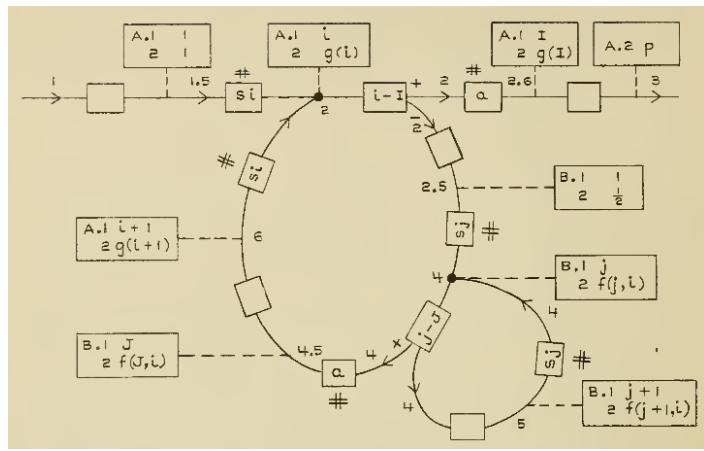


Figure 28. The first software flowchart [33: [vonNeumann1947](#)], figure 7.9, public domain assumed]

Architects and architecture are known for creating diagrams—abstract graphical representations of complex systems. The first known instance of applying graphical techniques to a digital problem was in 1947 [\[vonNeumann1947\]](#) and visual notations have evolved along with the field of computing ever since. Notable examples include:

- The IBM flowcharting template
- The Gane-Sarson data-flow diagram notation
- The Chen entity-relationship notation
- The Barker entity-relationship notation, including the "crow's foot" to indicate cardinality
- Harel state charts
- The Unified Modeling Language

and many more. (We touched on [data modeling](#) in Chapter 11.) We will examine Archimate, a current and widely used notation, in more detail in a future chapter section.



Figure 29. IBM flowcharting template [34: Image

<https://www.flickr.com/photos/mwichary/3249179483>, , downloaded 2016-10-10, commercial use permitted]

Research at Microsoft suggests that developers use diagrams for four purposes:

- Sharing
- Grounding (defining ambiguous interpretations)
- Manipulating
- Brainstorming

They argue "diagrams support communicating, capturing attention and grounding conversations [4]. They reduce the cognitive burden of evaluating a design or considering new ideas [13]" [\[Cherubini2007\]](#).

But visual notations have been problematic in the Agile community; as Fowler notes in his classic essay Is Design Dead:

[Agile method eXtreme Programming] de-emphasizes diagrams to a great extent. Although the official position is along the lines of "use them if they are useful", there is a strong subtext of "real XPers don't do diagrams".... I think the issue comes from two separate causes. One is the fact that some people find software diagrams helpful and some people don't. The danger is that those who do think that those who don't should do and vice-versa. Instead we should just accept that some people will use diagrams and some won't. [Fowler2004]

There is no question that some IT professionals, including perhaps some of the most skilled software engineers, find little of use in diagrams . As Martin Fowler says, "people like Kent [Beck, eXtreme Programming originator] aren't at all comfortable with diagrams, indeed I've never seen Kent voluntarily draw a software diagram in any fixed notation." However, it seems likely that Kent Beck and others like him are members of a programming elite, with a well-honed mental ability to process source code in its "raw" form.

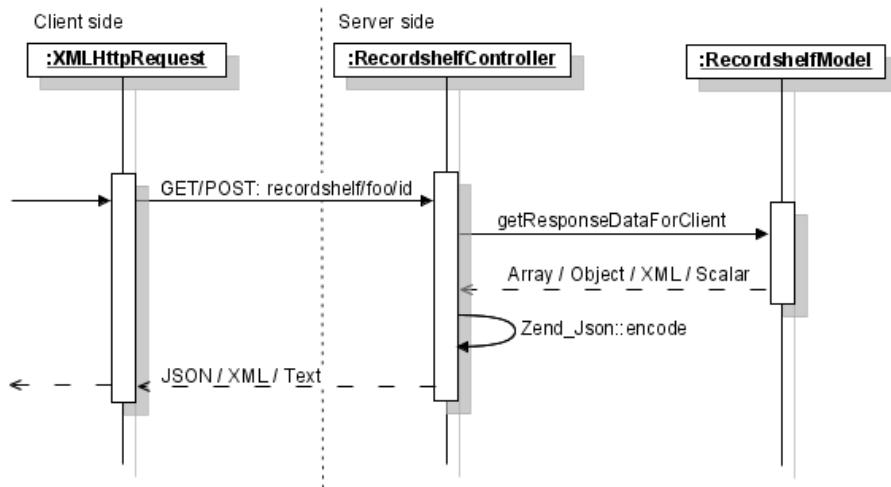


Figure 30. UML sequence diagram [35: Image <https://www.flickr.com/photos/raphaelstolt/514643232>, downloaded 2016-10-10, commercial use permitted]

However, if we're building systems to be operated and maintained by humans, it would seem that we should support the cognitive and perceptual strengths of humans. Because diagrams are more readily processed, they are often used to represent high level system interactions - how a given service, product or application is related to peer systems and services. Building such depictions can be helpful to fostering a shared mental model of the overall system objectives and context. The more complex and highly scaled the environment, the more likely one will encounter such artifacts as a means to creating the mental model.

The strength of human visual processing is why we will (probably) always use graphical representation to assist in the building of shared mental models. Specialists in the syntax and semantics of such designs will therefore likely continue to play a role in complex systems development and maintenance. Currently, if one seeks to hire such a specialist, one recruits some kind of architect - that is the professional role with the skills.

Note that flowcharts, data models, and other such diagrams tend to be associated more with the idea of "solutions" or "software" architecture. We'll cover the [architecture domains](#) in the next chapter section, including examples of business architecture diagrams.

Limitations of visualization

The big picture is part of the standard mindset of EA, which everyone immediately associates with the activities of an enterprise architect. However, many of these big pictures you meet in practice have been overabstracted to the point of insignificance and no longer address any relevant question. [Bente2012], p16.

— Bente et al, Collaborative Enterprise Architecture

Visualization has a number of limitations:

- It may be better suited for static structures than for dynamic processes,
- diagrams may have no real information content,
- diagrams are difficult to maintain, and there are diminishing returns the more they are elaborated and refined (e.g. for archival purposes)
- conversely, diagrams become less accessible the more complex they are,
- visualization can result in distorted understandings,
- ultimately, diagrams rely on deeper shared understandings that must be understood and managed.

Despite the familiarity of simple flowcharting, visual notations don't scale well in terms of representing program logic. Therefore, for dynamic or procedural problems, they tend to be used informally, as sketch or whiteboarding, or at the business analysis level (where the flowchart represents business logic, not detailed software). Dynamic processes also change more often than the static structures, and so must be updated more frequently.

More static structures, including data and class models and systems interactions, are still often represented visually and in the case of [data models](#) can be transformed from conceptual representations to physical schema.

However, any diagram, whether of a dynamic or static problem, can reach a level of density where it's no longer useful as a visual explanation. As diagrams become more complex, their audience narrows to those most familiar with them. Past a certain point they exceed the limits of human visual processing, and then are of little use to anyone.

For example, the diagram below, while intimidating, is likely useful to those who work with and study it. It would take some familiarization:

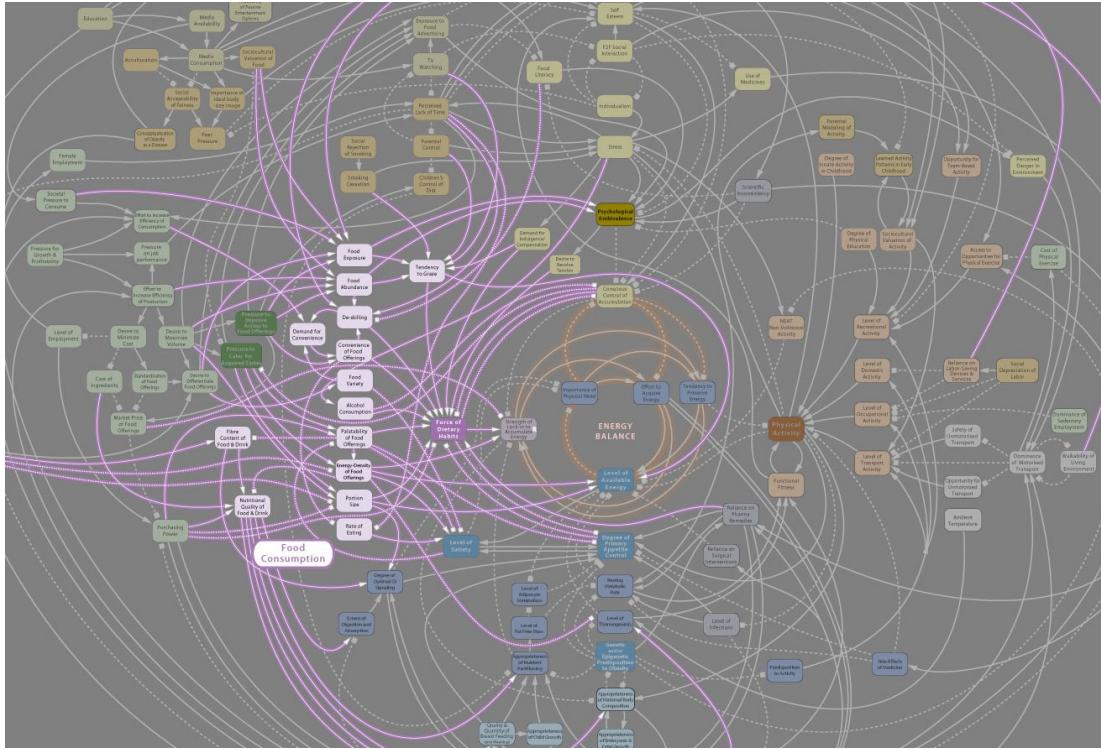


Figure 31. Complex diagram [36: Image <https://www.flickr.com/photos/pushandplay/2968259379>, downloaded 2016-10-10, commercial use permitted]

However, this next diagram is essentially unusable, as visually tracing any given line is too difficult, and it would be easy to mistakenly identify one bubble as dependent on another:

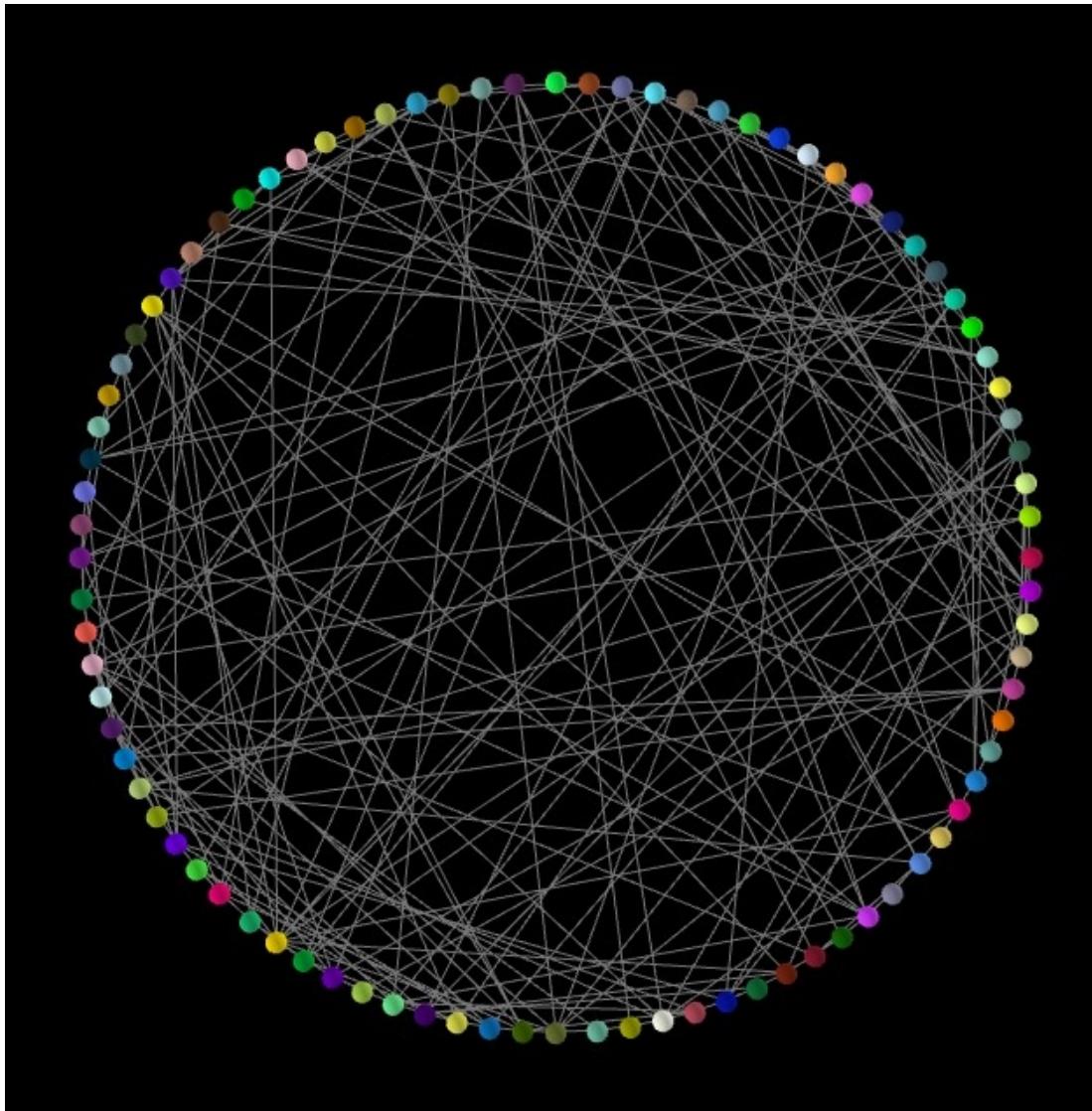


Figure 32. Another complex diagram [37: Image <https://www.flickr.com/photos/taedc/9614791576>, downloaded 2016-10-10, commercial use permitted]

This may seem like an obvious critique, but architectural diagrams of similar complexity and unusability have too often been produced.

Another issue with some diagrams is that they do not give a good sense of perspective or scale. This is sometimes seen in the business architecture practice of "capability mapping." formalisms don't give a good sense of scale. For example, suppose you see this diagram:

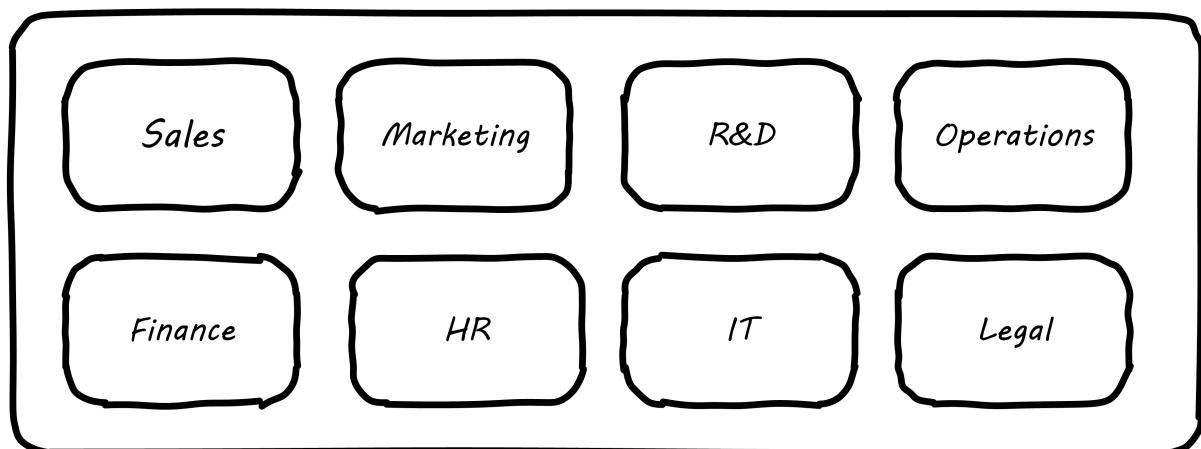


Figure 33. Simple capability map

Diagrams like this are common, but what does it mean that all the boxes are equally sized? Are there as many lawyers as sales people? Operations staff? It's not clear what the advantage is to putting information like this into a graphical form; no interactions are seen and the eight areas could more easily be expressed as a list (or "catalog" in the terms we'll introduce below). This brings us to the final problem listed above: visualizations rely on some **common ground** understanding. If boxes and lines are used for communication, their meaning should be agreed upon - otherwise, there is risk of mis-understanding, and the diagram may do more harm than good.

Regardless of the pitfalls, many architecture diagrams are valuable. Whether drawn on a whiteboard, in Powerpoint or Omnigraffle, or in a repository-based architecture tool, the visualization concisely represents a shared mental model on how the organizations will undertake complex activities. The diagram leverages the human preference for visual processing, accessing the powerful parallel processing of the visual cortex. Ultimately, the discussions and negotiations the architect facilitates on the journey to driving organizational direction are the real added value. The architect's role is to facilitate discussions by abstracting and visualizing powerfully so that decisions are illuminated and understood across the team, or broader organization.

Repositories and knowledge management

Artifacts are generally classified as catalogs (lists of things), matrices (showing relationships between things), and diagrams (pictures of things).

— TOGAF Version 9

The question was asked above, "why put things into a picture when a report is all that is needed?" We know that sometimes a picture is worth a thousand words, but not always. And sometimes the picture's components need more description than can conveniently fit on the actual diagram. This brings us to the topic of enterprise architecture as knowledge management.

Knowledge management is a broad topic, with a scope far beyond this book. But in the context of a digital organization, architecture can serve as an important component of an overall knowledge management strategy. Without some **common ground** of understanding, digital organizations struggle, and EA can help.

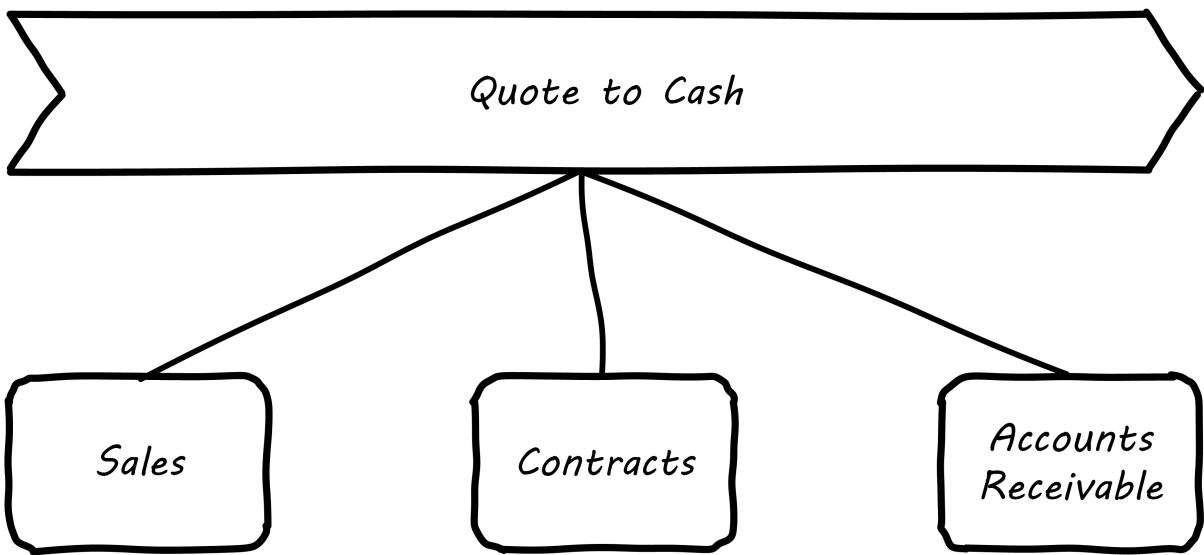
Catalogs, diagrams, matrices

As the quote from TOGAF above indicates, architecture can elegantly be represented as:

- Catalogs
- Diagrams
- Matrices

For example, consider this diagram:

Process and function diagram



It can be read as saying that the "Quote to Cash" process depends on the following functions:

- Sales
- Contracts
- Accounts Receivable

Notice that the following matrix can be read in the same way:

Process and function matrix

Catalog of functions

Catalog of processes	Sales	Contracts	Accounts Receivable	Vendor Management	Accounts Payable	Human Resources	Information Technology	Payroll	Benefits
	Quote to Cash	X	X	X					
Procure to Pay				X	X				
Hire to retire						X	X	X	X

"Quote to cash," which appeared as a chevron in the diagram, is now one of a list:

- Quote to Cash
- Procure to Pay
- Hire to Retire

This list can be called a "catalog." Similarly, there is another catalog of functions:

- Sales
- Contracts
- Accounts Receivable
- Vendor Management
- Accounts Payable
- Human Resources
- Information Technology
- Payroll
- Benefits

The functions appeared as rounded rectangles in the diagram.

There are pros and cons to each approach. Notice that in about the same amount of space, the matrix also documented the dependencies for two other processes and six other functions. The matrix may also be easier to maintain; it requires a spreadsheet-like tool, where the diagram requires a drawing tool. But it takes more effort to understand the matrix.

Maintaining a catalog of the concepts in a diagram becomes more and more important as the diagram scales up. Over time, the IT operation develops significant data by which to manage itself. It may develop one or more definitive portfolio list, typically applications, services, assets, and/or technology products. Distinguishing and baselining high quality versions of these data sets can consume much resource, and yet managing the IT organization at scale is nearly impossible without them. In other words, there is a [data quality](#) issue. What if the boxes on the diagram are redundant? Or inaccurate? This may not matter as much with a tight-knit team working on their whiteboard, but if the diagram is circulated more broadly, the quality expectations are higher.

Furthermore, it is convenient to have data such as a master lists or catalogs of processes, systems, functions, or data topics. We might also want to document various attributes associated with these catalogs. This data can then be used for operational processes, such as [risk management](#), as we have discussed previously. For these reasons and others, enterprise architecture repositories emerge.

Architecture data management

When we establish a catalog of architectural entities, we are engaging in [master data management](#). In fact, the architectural concepts can be represented as a form of [database schema](#):

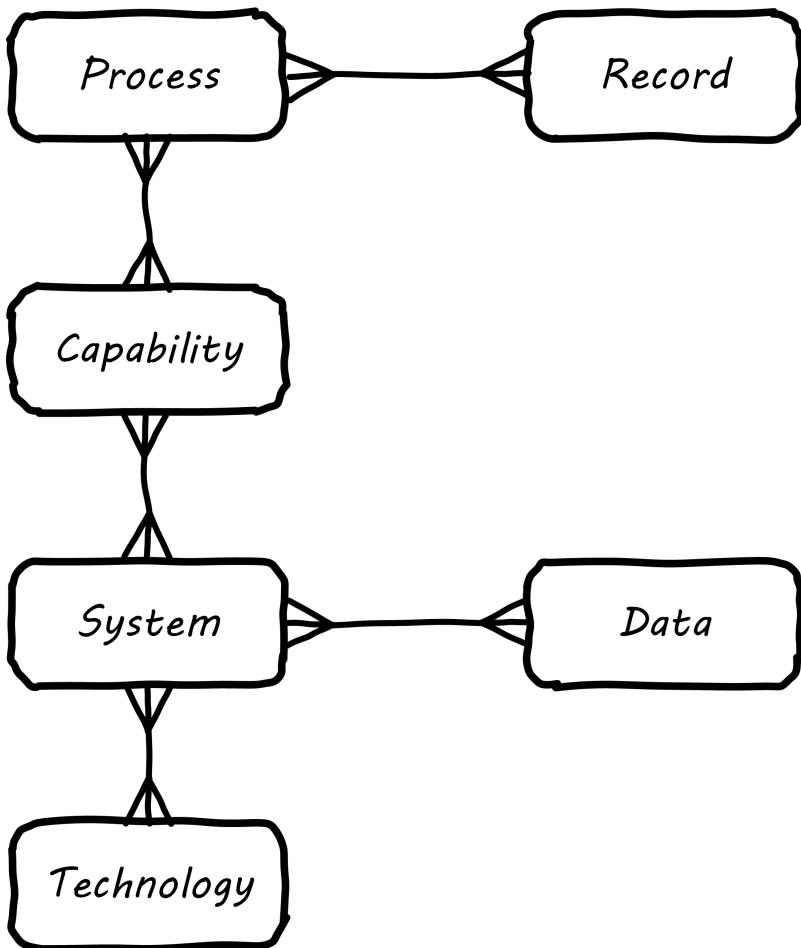


Figure 34. A simple metamodel

NOTE

A data model that organizes data about data and its related systems can be called a *metamodel*.

Thus, material that we first saw in diagram form can be stored in a database. Systems that enable this are called enterprise architecture repositories.

Architecture repositories require careful management. A common anti-pattern is to acquire them without considering how the data will be maintained. The concepts in the repository can be subjective, and if it is intended that they be of high [data quality](#), investments must be made. Some kind of registration process or decision authority must exist for the creation of (for example) a new, official "system" record. Misunderstandings and disagreements about the very meaning of terms like "system" or "technology." (We discussed some of the general issues in Chapter 11, with the [ontology problem](#)). Such issues are especially difficult when enterprise architecture repositories and metamodels are involved. Frequent topics:

- Is an "application" different from a "service"? How?
- What is the relationship between a "capability" and a "function"? Or a "capability" and a "process"?
- How can we distinguish between "systems" and "technologies"?
- What is the relationship between a "product" and a "service," especially if the service is a market-facing digital one?
- What is the relationship between:

- Value chain
- Value stream
- Process
- Activity
- Task

And so on. One might expect that there would be industry standards clarifying such issues, and in some cases there are. In other cases, there are either no standards, or the standards are obsolete or conflicting.

Finally, there are a number of other systems that may interoperate with the architecture repository. The most important of these is the configuration management database (CMDB) or system (CMS) that underlies the IT service management tooling. These tools also need to know at least about systems and technologies, and may be interested in higher level concepts such as business capability. However, they usually do not include sophisticated diagramming capabilities or the ability to represent a system's future state.

Other tools may include project management systems, portfolio management systems, risk management systems, service level management systems, and others. Application and service master data in particular is widely used, and if the EA repository is [System of Record](#) for this data there will be many outbound interfaces.

An economic view

NOTE

The discussion below also applies to the Configuration Management Database (CMDB) as well as other similar repositories.

Part of the challenge of any repository is what data to manage. How do we think more systematically about this? First, we need to understand why we want to assemble this data in a ready-to-query repository. There are two major reasons why we store data:

- There are no other sources for it. If we don't establish a system of record, the data will go unmanaged. We won't know what servers we have, or what applications we are running.
- There may be other sources for the data, even systems of record. But we need an operational data store to bring the various data sources together in a way that makes them more efficient to query.

For either kind of data, you need to have an economic understanding of why you want it. Suppose you need to find out what applications you are running, because you want to rationalize them. You could invest weeks of research into the question, costing perhaps tens of thousands of dollars worth of yours and others' time, to create a one-time spreadsheet.

But what happens when there are multiple purposes for the data? You find out that the security group also wants a master list of applications and has been compiling a different spreadsheet, for example. What happens when the same engineers and managers are asked for the same data over and over again, because there is no repository to maintain this organizational memory?

The challenge is, when does it make economic sense to pre-aggregate the data? The following

economic graph may help:

The graph may be familiar to those of you who studied economics. On the left, you have the assumption of no architecture repository, and on the right you have a comprehensive architecture repository. With a less comprehensive architecture repository, you are paying some cost in research and outage impacts. You also are incurring more risk, which can be quantified. On the other hand, with a comprehensive architecture repository, you incur more costs in maintaining it. You need processes that have direct cost to operate, as well as imposing indirect costs such as cost of delay (e.g. if updating the architecture repository slows down the release schedule).

But in the middle is a sweet spot, where you have “just enough” architecture repository data. This optimal architecture repository scope represents the real savings you might realize from instituting the architecture repository and the necessary processes to sustain it. This is not a complete business case, of course. Your projected savings must be offset against the costs of acquisition and operations, and the remaining “benefit” needs to exceed your organization’s hurdle rate for investments.

A simple example:

Let’s say that you have identified a set of use cases indicating \$250,000 maximum savings from an accurate and optimally scoped architecture repository. The implementation is projected to cost \$200,000 over 3 years, for net benefits of \$50,000. This means that you have an ROI of 25% (\$50k/\$200k). You can’t get 25% return on investment in the stock market, so the CFO sees this as a good use of money. On the other hand, if you only are projecting \$15,000 in net benefits, for an ROI of 8%, the CFO may well say, “I can do better with other projects, or by leaving our money in the stock market.”

Of course, estimating benefits such as reducing redundant research are not simple. The book *How to Measure Anything* by Doug Hubbard [Hubbard2010] is a very useful resource for these kinds of problems. You may need to consider using techniques such as Monte Carlo Analysis for business benefits that are more probabilistic. But you do not need to throw up your hands and say it’s all just “intangible,” successfully making challenging business cases of this nature is possible.

The "rationalization" quest

A foolish consistency is the hobgoblin of little minds.

— Ralph Waldo Emerson

"Rationalization" is often listed as one of the major outcomes of enterprise architecture. What is meant by this? Let’s return to our scenario of [one company acquiring another](#). As the newly merged company takes stock of its combined assets, it becomes clear that decisions need to be made. Among other areas, redundant systems exist for:

- Marketing
- Human resources
- Accounting

The digital pipelines also are inconsistent, one being based on Github and Travis CI, the other being based on local git and Jenkins.

Decisions need to be made as to which systems will be "go-forward." While the teams involved will have strong input into the system decisions that affect them, and will do most of the work, there is concern that some overall view and coordination of the effort is required. What if teams cannot come to consensus? What if there is an opportunity to save money by standardizing on one vendor to support multiple diverse teams? For these reasons, the company assigns an architect to work closely with the overall merger program.

A merger is a dramatic example of a rationalization scenario. Established, ongoing companies, even without mergers, find that redundancy tends to accumulate. This is a normal outcome of the [innovation and commoditization cycle](#); when technologies are new, organizations may experiment with several providers. When they become more standardized and commodity, the desire for efficiency drives rationalization.

One of the challenges for rationalization is whether the economics and business context of any given rationalization effort are well understood. Consistency as an end in itself is not necessarily valuable. The impact on enterprise value must be established: will the organization actually benefit from improved vendor leverage, operational integration, or a reduced security attack surface? If not, perhaps seeking "rationalization" is not the best use of organizational resources.

We close this section with some case studies on rationalization.

Application rationalization

One core question decided by governance is how much autonomy is granted to business units or geographical regions. In case this autonomy is high, would a quest for high IT integration and standardization not be like fighting windmills? [\[bente2012\]](#) p.45

— Bente et al, Collaborative Enterprise Architecture

A large electronics retailer purchases a smaller chain of stores specializing in vinyl records and CDs, just as the market for these are declining. The decision is made to integrate all the systems: Point of Sale, inventory, HR & Payroll, etc, in the interest of efficiency and rationalization.

Work progresses on this effort for nearly a year, and then the surprising news is announced: the newly acquired company is to be sold off! All the work that went into rationalizing the systems is wasted, and the independent operating model for the smaller chain has to be completely restored. Clearly, in this case, rationalization was not rational.

Data and information

A large financial institution had allowed its computer systems to become fragmented. Acquisitions and independence among the lines of business meant that if a customer moved, they might need to contact customer service repeatedly to change their address. This was a failure of [master data management](#); there was no clear system of record for the customer address.

While we first discussed this kind of issue in Chapter 11, solving the problem requires more than data architecture. Establishing one data source as "master" requires considerable engineering in such an environment: there are capacity, security, interface, and business process concerns to work

through, that in this particular case took tens of millions of dollars of investment, in response to increasing customer unhappiness.

And once a system of record is established and the data interfaces have been constructed reflecting its status, its role needs to be preserved. Independent teams may once again start to master their own customer data, "just for now," because perhaps accessing the system of record seems to be too much work. Avoiding such scenarios is why some architecture organizations institute project reviews.

Technology rationalization case #1



Figure 35. Airbus A-380 [38: Image credit <https://www.flickr.com/photos/44400809@N07/4081438180>, downloaded 2016-10-11, commercial use permitted]

In 2006, Airbus was in the midst of developing the gigantic A-380 aircraft. Because of the scale of the effort, multiple teams of engineers in different countries were working on the effort. As William Ulrich says of the German and French teams, "having failed to coordinate efforts to wire the world's largest plane, they were surprised to learn that they had used different approaches to designing the A380's wiring system" [Ulrich2010], p. 32. This was because the teams were using different software for wiring design of the critical wiring design software [Clark2006]. The use of different software to fulfill the capability resulted in a year's delay, a \$4.5bn order loss to Boeing, and a 26% drop in Airbus stock.

Technology rationalization case #2

Henrik Kniberg worked on the PUST project for the Swedish national police, which was a project to deploy laptops to the Swedish police. Because of its urgency, it was allowed to use Lean and Agile techniques, and received favorable media coverage and was generally deemed a success [Kniberg2011].

Unfortunately, a drive for technology "rationalization" resulted in the re-platforming of the system on a commercial platform (Siebel) [Kniberg2013]. Furthermore, an [open loop waterfall](#) method was used to deliver the system. Kniberg reports that the result has been an unsuccessful rollout, unhappy users, and cost to the Swedish police department of about 1 billion pounds (\$1.6B).

As we have discussed, there can be an economic case for rationalization. It reduces support costs and increases vendor leverage. However, 1 billion pounds would have bought a lot of support & vendor leverage for the "nonstandard" technologies. It is hard to see that this was an economically rational decision, especially given the particular risks of police work (the system in question is used

by officers in the field, to keep records on criminal cases.) It does not take much imagination to think of scenarios where a difficult to use system for a police officer could have costly or even tragic results. In this case, the quest for rationalization appears misguided.

Service or technology rationalization?

A large US company with a robust architecture program found itself with a serious controversy. A request had been made of the Technology Lifecycle Management process for the Subversion version control system to be approved for use by development teams. A central version control team (what might now be called a Release Engineering team) was opposed to this, citing duplication of technologies.

Multiple version management systems already did exist, as different platforms had different requirements. In general, the thinking was that the approval process should simply verify that the tool was "fit for use" and not worry about redundancy.

However, as the discussion escalated, it became clear that the central release engineering team was worried not so much about the technical capabilities of Subversion, but rather a "loss of franchise" for their services - they had been established as the primary shared service for source control. Many teams did use their shared services, but there was also a regulatory issue - they were seen as an authoritative and auditable repository by regulators.

So the situation was complex; was the company "rationalizing" at the technology or service level? Ultimately, a solution (unsatisfactory to most) was arrived at that the Subversion-using team still had to use the central source repository as a system of record. They could develop locally using Subversion, but were still expected to archive their source in the central tool.

In the next chapter section we will go into a more thorough discussion of the service/application "level" versus the technology "level."

Architecture domains

In the last section, we discussed what architects **do**. The focus was primarily architects in a "team of teams" environment, not strictly at the level of a single product team. However, while this chapter has a strong Enterprise Architecture orientation, it is also about the practice of "architecture" more generally. In this chapter, we will break down the different practices of architecture and their relationships.

We have seen the [Zachman Framework](#) previously. The higher levels are considered "business" or "operating model" concerns, while the lower levels are more technical. In discussing the various domains of architecture, however, a simpler structure is useful. The numerous columns in the Zachman framework don't necessarily translate to specific architecture domains (for example, there are many data architects representing the "what" column, but not many who specialize strictly in questions of timing — the "when" column.) Similarly, we can simplify the number of rows by consolidating them into three.

Archimate is a modeling language and standard of The Open Group, which we will discuss more later in the chapter. It uses a framework that can be viewed as a simplification of the Zachman

model:

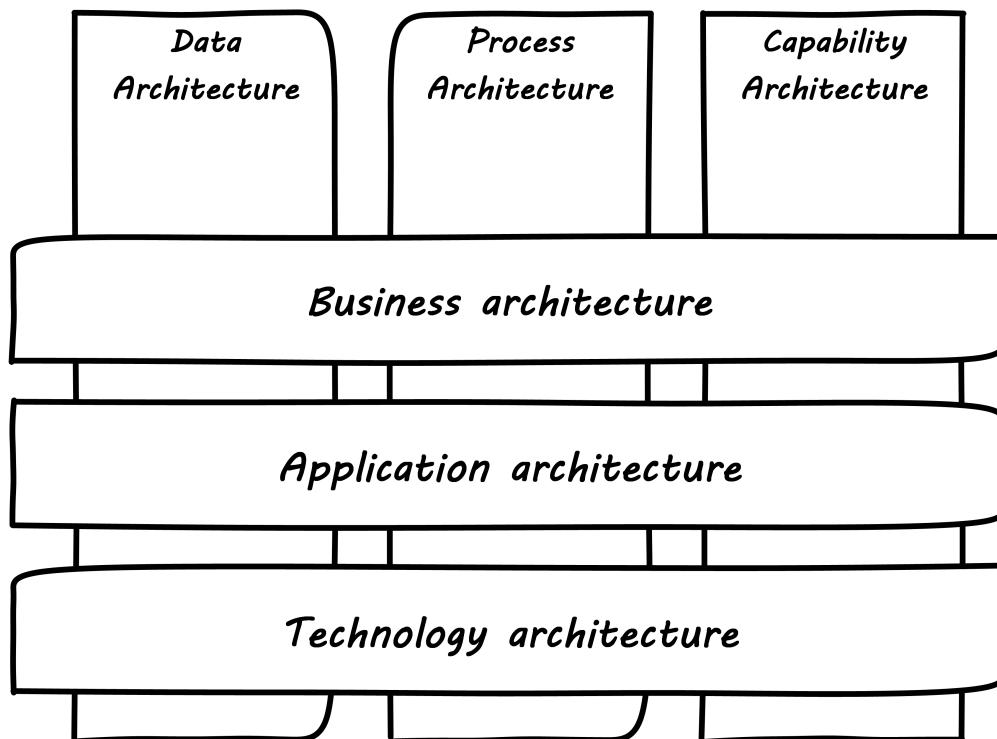


Figure 36. Similar to [Open2012], section 2.6

As we look at the overall structure of the architecture disciplines, we have three disciplines that correspond to the columns. We'll call these "perspectives":

- Data Architecture
- Process Architecture
- Capability Architecture

And three disciplines that correspond to the hierarchical layers:

- Business architecture
- Application architecture
- Technology architecture

Does this mean that we have nine flavors of architecture, one for each intersection? Not necessarily. Some intersections make more sense than others, and some tend to merge with their neighbors.

For example, see the Data Management Body of Knowledge [DAMA2009], which covers the gamut of information and data topics from high level glossaries all the way down to physical database administration. Data architects can and do work across all levels in their perspective.

Architecture perspectives

Data architecture

From top to bottom, the data architecture ("what") perspective includes concepts such as :

- The universe or domain of discourse
- Ontologies, semantics, and conceptual models
- Logical data models
- Data subjects & records
- Classes
- Entities/attributes/relationships
- Tables/columns/constraints

We covered data architecture in depth in [Chapter 11](#).

Process architecture

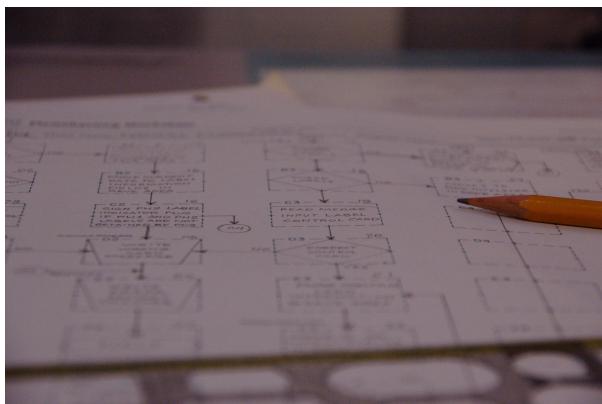


Figure 37. A flowchart [39: Image credit <https://www.flickr.com/photos/lespetitescases/16715406524>, downloaded 2016-10-16, commercial use permitted]

Process architecture is concerned with why and how activities are performed. It includes the detailed, step by step understanding of activities, in a transparent way. From top to bottom, it includes concepts such as:

- [Value chain](#)
- Value stream
- Business process
- Algorithm
- Workflow
- Activity
- Procedure
- Task
- Step

Importantly, [processes CROSS organizations](#). An "onboard employee" process, as we have seen, may require participation from multiple organizations. We covered process management in depth in [Chapter 9](#).

Capability architecture

The last column represents steady state activities. "Hire employee" is a process; "Manage Human Resources" is a capability. We do not necessarily know all the steps or details; we just know that if we ask the function or capability for some result, it can produce it. This perspective includes:

- Function and its relatives
- Function
- Capability
- Service (sometimes)

We defined function previously in [Chapter 9](#). Note that there is little consensus (and as of 2016 much industry debate) around whether functions are the same as capabilities; this textbook sees them as at least similar.

Capability is an important concept in business architecture, as it has emerged as the preferred concept for investment. We do not invest in data, or process, except as they are realized by a supporting capability.

A comprehensive graphical depiction of "capabilities" may be used to help visualize portfolio investments, sometimes using green/yellow/red color coding - this is called "capability heat mapping."

Architecture layers

Business architecture

Business architecture offers views of the business that are unavailable from other sources, including IT. Business architecture can tell you what is being done, by which business units, for certain customers, involving various products, via certain processes, involving selected business information. Business architecture generated blueprints serve as the basis for root cause analysis of critical business requirements while providing the foundation for establishing a solution-oriented roadmap that leaves the speculation and guesswork by the roadside. In a word, business architecture delivers "transparency" to a wide variety of internal teams, roles and business units [\[Ulrich2010\]](#), p. 195.

— William Ulrich, *Business Architecture: The Art and Practice of Transformation*

Business architecture is defined by the Business Architecture Body of Knowledge (BIZBOK) as "a blueprint of the enterprise that provides common understanding of the organization and is used to align strategic objectives and tactical demands."

BIZBOK goes on to say that "the value of business architecture is to provide an abstract representation of an enterprise and the business ecosystem in which it operates. By doing so, business architecture delivers value as an effective communication and analytical framework for

translating strategy into actionable initiatives. The framework also enhances the enterprise's capacity to enact transformational change, navigate complexity, reduce risk, make more informed decisions, align diverse stakeholders to a shared vision of the future, and leverage technology more effectively."

BIZBOK covers the Osterwalder Business Model Canvas extensively. ([BAGuild2016], pp. 282-297.) In so doing, it clearly implies that the concept of the business model is of interest for business architects. Because of this, it's helpful to view **business architecture as the component of enterprise architecture most concerned with the business model, in addition to the operating model.**

More specifically, there are a number of concerns that business architecture includes:

- Value streams
- Capabilities
- Organization
- Information
- Stakeholders
- Vision, Strategies, and Tactics
- Initiatives and Projects
- Decisions and Events
- Metrics and Measures
- Products and Services
- Policies, Rules, and Regulations

from [BAGuild2016] p. 2. The reader might notice some overlap with COBIT enablers, which also include Information, Policies, and Organization.

On the other hand, we DO NOT expect to see in business architecture the following:

- Specific technology products (Oracle 11g)
- Software architectures (design patterns, class models, etc)
- Detailed deployment diagrams
- Specific project plans
- Detailed flowcharts
- Specific devices

Application architecture

Application, or application system, like data, process, and capability, is a fundamental and widely used architecture perspective, as well as a layer. It can be defined as ""a fixed-form combination of computing processes and data structures that support a specific business purpose." [Betz2011], p. 125. An application system is practically relevant, obtainable and operable. (You can buy, or realistically build, one of these.)

Application architecture can have two meanings:

- The architecture of a given application
- The architecture of application interactions

For this book, we'll leave the architecture of a given application for solutions and software architecture. Application architecture is the interaction of multiple applications (which may include digital products and/or services, depending on organization terminology). In a complex, multi-product environment, application architecture tends to focus on the interfaces and interactions between the application systems. It's often a concern when systems are considered for retirement or replacement (for example, when a comprehensive ERP solution is brought in to replace several dozen home-grown applications.)

Application architecture is also concerned with the [application lifecycle](#), as covered at the start of this section.

Technical architecture

Where business architecture intersects with the business model, technical architecture overlaps with actual engineering and operations. In particular, technical architecture tends to be concerned with:

- Identification of new technical platform capabilities: for example, does the organization need to bring in a NoSQL platform? Private cloud?
- Choice of vendor products, once a technical need is established
- Establishing infrastructure services as appropriate
- Defining appropriate usage, including infrastructure design patterns
- Tracking the [lifecycles](#) of the selected products and dependent services, and making appropriate plans

Other forms of architecture

There are other kinds of architecture that don't fit neatly into this arrangement:

- Solutions architecture
- Software architecture
- Information architecture (UX-related definition)

Solutions architecture

Solutions architecture especially is a loose term. In general, it is restricted to one product, or a few products working together, as a "solution" to a business problem. Within that scope, it may incorporate concepts from infrastructure to business architecture.

Software architecture

Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.

— Grady Booch

"Architecture" in the sense of pure software is a topic with much research and writing. In this book, it has been a concern since Section 1, and so we don't talk as much about it here.

Software architecture is usually aligned with application architecture, but not all application architecture is software architecture; application architecture may also include packaged solutions whose internal architecture is not a concern.

Information architecture (alternate usage)

Information architecture may mean the higher, more business-relevant levels of data architecture. However, the term also is used in relation to application architecture, in the sense of how the user understands the meaning and data represented by a web site or application, or even just the navigation structure of a Web site.

Agile and architecture

...we encountered companies that, despite having a fully institutionalized EA in place, were in a state close to paralysis...Although EA has reached the mainstream, a skeptical undertone with regard to its effectiveness has always existed. (p 12-13)

— Bente et al, Collaborative Enterprise Architecture

The relationship between architecture (both "enterprise" and other forms of architecture) and current Agile, DevOps, and digital product development approaches is too often troubled. However, the hope is that this book has given you a set of tools for resolving these concepts in a productive way.

This chapter will challenge you by presenting the polemical arguments directly and frankly, as two sides to a debate, concluding with thoughts on finding common ground. Note that these points of view are deliberately extreme, to make the point:

Author's Note

As a practicing architect I can confirm that there is often friction between developers with an Agile perspective and architects, whether "enterprise" or other kinds. This carries through into industry discussions and conference presentations.

I think the best approach to resolving this conflict is through careful understanding of the perspective of each "side." Both have validity. I recommend framing the conversation in terms of [architecture's value impacts](#) and quantifying wherever possible.

The hubris of architecture

The goal of enterprise architecture is to act as a guide, perhaps a pathfinder, who takes the enterprise on a transformational journey—from an incoherent and complex world with line-of-business separation, product-specific stovepipes, legacy systems estate, and costly operation to a more rationally organized and useful state with multiservice, revenue-generating platforms and an efficient operational regime. On the way, radical surgeries may be required to eliminate duplication, reduce costs, improve reliability, and increase agility in the business. EA acts as a strategic foundation for business enablement. [Bente2012] p9

Product development organizations often experience architecture and its goals as unwarranted interference, imposing a high [Cost of Delay](#) with little apparent return on investment. Architecture approvals can be required on:

- application designs
- database designs
- selection of technology products

and other such topics. When development cannot proceed without those approvals—or if the approvals come at the cost of expensive re-work—the experience can often be challenging. Bente et al warn: "if enterprise architects claim to be the only decision-making body in technical matters, there is a huge risk that they create a bottleneck...The practical consequence is that projects deliberately circumvent the enterprise architects..." [Bente2012] p19

And, looking more broadly at the practice and history of IT architecture, the case against it is strong. Enterprise architecture has presented itself as a solution to complexity, long IT time scales, business frustration, and other various IT problems. These issues are at this writing being solved, but not by architecture—at least not visibly. Instead, visible and publicized progress has come through the increasing adoption of Agile and DevOps practices rethinking open-loop, slow feedback, batch-oriented delivery. Architecture has failed in many ways:

It failed to realize the emergent issue of too much enterprise work in process, instead championing the proliferation of enterprise processes and their associated queues. Fundamentally, they did not adopt enough of a systems view of IT delivery. Architects' motivation for "efficiency" and interest in capability mapping did not help the cause of cross-functional teams. Instead, functional silos were reinforced as supply-centric "capabilities" and the antipattern of "bringing the team to the work" were promoted as enterprise standard operating procedure - despite the growing evidence of Scrum and Agile success. The iterative, experimental narrative of Lean Startup did not originate from EA.

Despite a professed interest in systems theory, architecture has only recently begun to realize the fundamental problems of stage-gated delivery, big bang releases, and so forth. Architecture gap analysis usually assumed that project investment would be used to fill them. Architecture often merited the criticism of "top down planning," which in complex systems domains too often doesn't work. Architects too often fall into the trap of the [HiPPO](#) (Highest Paid Person's Opinion). A sense of [Lean Startup](#) experimentation, of placing bets on options and testing hypotheses, is not part of the mainstream enterprise architecture understanding. Instead, the architecture is presented as an established fact, with "governance" to ensure conformity. Hypothetical "synergies" emerging from "common platforms" are often offered as justification for architecture, with little follow up in

measuring actual value delivered.

Justifications for architecture often invoke "complexity" in the portfolio of systems. In response, architecture has often given in to the desire for a complete "radical surgery" systems re-engineering, the temptation of the "clean slate." But as Jez Humble accurately notes,

A common response to getting stuck in a big ball of mud is to fund a large systems replacement project. Such projects typically take months or years before they deliver any value to users, and the switchover from the old to the new system is often performed in "big bang" fashion. These projects also run an unusually high risk of running late and over budget and being canceled. Systems rearchitecture should not be done as a large program of work funded from the capital budget. It should be a continuous activity that happens as part of the product development process. [Humble2013] ch10

Architecture methodology, with its focus on identifying capability gaps for feeding into the project portfolio process, has perhaps been too prone to supporting these large, troubled programs. As we know from our earlier chapters, large system changes are inherently risky and any intervention into a complex system is better undertaken as a series of smaller, incremental changes with frequent monitoring and assessment.

The hubris of Agile

Instead of tapping into the existing knowledge of the organization the autonomous team is prone to reinvent the wheel, and the wheel that they reinvent will not always be superior to the one we are currently using [Reinertsen1997], p. 104.

— Donald Reinertsen, Managing the Design Factory

The Agile community has its own blind spots and challenges. Speed is seen as a good in itself, too often without an economic model. Agile teams often clash with enterprise governance processes that have sound compliance and financial benefits. Phrases like "you aren't gonna need it" are used to justify lapses of due diligence on critical capabilities, and standard platforms and vendors are seen as unreasonable limitations on team autonomy—to the point where it seems some teams' interest is primarily in padding their resumes with as many new technologies as possible, regardless of the long-term consequences for the organization.

The limitations of Cost of Delay

When it comes to system implementation, the temptation to be fast, often under the nom de guerre of agile, can soften quality controls and threaten product usability, reliability, safety, and lifecycle cost [Malan2010].

— Ruth Malan

[Cost of Delay](#) is a real and often overlooked issue, in understanding the net value of architecture. But it is only a factor, and does not eliminate the value proposition of architecture. If the Cost of

Delay is only a few hundred dollars a month, but the risk or technical debt represent millions, then delay may be appropriate. Don Reinertsen, who has done more than anyone to promote the idea of Cost of Delay, emphasizes that **all** decision making must take place within an economic framework ([\[Reinertsen2009\]](#), chapter 2) and that means that the other **architectural impact factors** on organization value must also be considered.

Documentation

Documentation has been a core concern of the Agile movement, being mentioned in one of the four core principles of the Agile Manifesto:

"Working software over comprehensive documentation" [\[Alliance2001\]](#).

When documentation primarily takes the form of **secondary artifacts**, it is appropriate to question the need for it. "The code is the documentation," some will argue. While it is true that good coding practices result in easier-to-understand (and maintain) source code, the code cannot be the only documentation. However, as Ruth Malan notes,

...for systems of sufficient scope and complexity to warrant teams (of teams) working on (incremental) implementation and evolution, the sheer mass of code can make it hard to discover the essential structure from bottom-up decisions made entirely through the medium of code. [\[Malan2010\]](#)

In terms of systems theory, a complex software system has emergent behavior, not obvious from just looking at its components. Because the system's behavior can't be reduced to its pieces, "self documenting code" can only go so far. The behavior of the assembled components as a system needs to be represented somehow, in a way that transcends the mere mechanics of the pieces. Abstraction is necessary to understand emergent behavior, and this leads inevitably to **visual representation**. Without some attention to documenting overall context and systemic intent & behavior, the effectiveness of the overall human/computer system degrades. For example, Alistair Cockburn reports that the Chrysler Comprehensive Compensation project, one of the first widely-reported Agile projects, was eventually halted, and

...left no archived documentation ... other than two sentence user stories, the tests, and the code. Eventually, enough people left that the oral tradition and group memory were lost [\[Cockburn2007\]](#), pp. 41-43

In short, failure to sustain a shared mental model of a complex system is a risk that may result in loss of that system's value.

Sourcing and technology standards

Agile and DevOps are software-development centric, and have transformed that world. However, digital organizations don't always build everything. There is a complex web of supplier relationships even for organizations with robust software development capabilities, and many organizations would still prefer to "buy rather than build." Software may be eating the world, but that doesn't mean everyone employs - or should employ - software developers. Agile has not had a primary focus on **sourcing**, and certainly evaluating commercial software is not a common topic.

Suppose you have an idea for a digital product, and you know that you will be (at least in part) assembling complex services/products produced by others? Suppose further that these provided

services overlap (the providers compete)? You need to carefully analyze which services you are going to acquire from which provider. You will need a strategy, and who is it that analyzes these services and their capabilities, interfaces, non-functional characteristics, and makes a final recommendation as to how you are going to bring them all into one unified system?

It is easy to say things like, "the teams get to define their own architecture" but at some point the enterprise must reckon with the cost of an overly diverse supplier base. This is a very old topic in business, not restricted to IT. At the end of the day, supplier and sourcing fragmentation costs real money. Open source, Commercial-off-the-shelf, Cloud, in-house... the options are bewildering and require experience. In a sense, the supplier base itself is an inventory, subject to aging and spoilage. (We can consider this another way of understanding technical debt.) A consistent evaluation approach is important (preferably under an economic framework Reinertsen/Hubbard). And at some point, product development teams should not have to do too much of their own R&D on possible platforms for their work.

Architecture as emergent

"At Netflix, we had no central control [of the architecture] ... The goal of architecture was to create the right emergent behaviors..." [\[Bloomberg2014\]](#)

— Adrian Cockcroft, former CTO Netflix

The Agile Manifesto is well known for saying "The best architectures, requirements, and designs emerge from self-organizing teams" [\[Alliance2001\]](#). This is one of the more frequently discussed Agile statements. Netflix CTO Adrian Cockcroft has expressed similar views (quote above).

A key question is whether "architecture" is considered at the single product or multi-product level. At the single product level, collaborative teams routinely develop effective software architectures. However, when multiple products are involved, it is hard to see how all the [architectural value](#) scenarios are fulfilled without some investment being directed to the goals of cross-product architectural coordination. It helps when rules of the road are established; both Amazon and Netflix have benefited from having certain widely accepted platform standards, such as "every product communicates through APIs." Netflix has had a long term commitment to Amazon Cloud services; it is probably not acceptable for teams there to decide on a whim to deploy their services on Google Compute Engine or Microsoft Azure, so at least in that sense Netflix has an architecture. The question gets harder when layered products and services with [complex lifecycle interactions](#) are involved.

Microservices can reduce the need for cross-team coordination, but as we [previously discussed](#), coordination needs still do emerge.

Towards reconciliation

So how do we reconcile Agile with architecture practices, especially enterprise architecture and its concerns for longer lifecycles, aggregate technical debt, and governance? We need to understand why we look to architecture, what utilizing it means, and how it ultimately adds value, or doesn't, in the organization.

Why: Creating the context

One principle throughout this book has been "respect the team," because true product value originates there. If teams are constantly fragmented by enterprise operating models and governance mandates, their ability to creatively solve business problems is hampered. Command and control replaces emergence, motivation declines, and valuable creativity is lost. **Enterprise architecture must first and foremost protect the precious resource that is the high-performing, collaborative, creative team.** As we've discussed, imposing multiple governance checkpoints itself [adds risk](#). And while it's inevitable that the team will be subject to organization-wide mandates, they should be given the benefit of the doubt when autonomy collides with standardization.

When enterprise architecture takes on true business architecture questions, including how digital capabilities are to be enabled and enhanced, Agile insights become an input or kind of requirement to business architecture. What capabilities require high-performing, cross-functional teams? What capabilities can be supported with project-based temporary teams? The more valuable and difficult the work, the more it calls for the careful development of a common mental model among a close-knit team over time. Driving organizational capability investment into long-running team structures becomes a strategy that organizational architects should consider as they develop the overall organizational portfolio.

Architecture adds value through constraining choices. This may seem counterintuitive, but the choice is often between re-using a known existing platform, or engaging in risky research and development of alternatives. R&D costs money, and itself can impose delay on establishing a reliable digital pipeline.

But ultimately, the fundamental objective remains customer and product discovery. All other objectives are secondary; without fulfilling customer needs, architectural consistency is meaningless. Optimizing for the fast creation of product information, tested and validated against operational reality, needs to be top of mind for the architect.

What: the architecture of architecture, of the digital pipeline itself

The digital pipeline ultimately is a finely tuned tool for this creation of information. It, itself, has an architecture: business, application, and technical. It operates within an economic framework. To understand the architecture of the digital pipeline is in a sense to understand the "architecture of architecture."

As we've discussed above, architecture, like [staff functions](#) generally, is in part a coordination mechanism. It collects and curates knowledge and sustains the organization's understanding of its complex systems. Architecture also identifies gaps and informs the investment process, in part through collecting feedback from the organization.

If architecture's fundamental purpose is enabling the right emergent behavior, there are still questions about how it does so. Architecture adds value in assisting when:

- systems are too big for 1 team
- features are too complex to be implemented in 1 iteration
- features require significant organizational change management

As a coordination mechanism, it can operate in various ways including planning, controlling, and collaborating. Each may be appropriate for a given challenge or situation. For example, different approaches are required depending on whether the product challenge is *Flower* or *Cog*. A flower is not engineered to fill a gap. A cog is. Market-facing experiments need leeway to pivot, where initiatives intended to fill a gap in a larger system may require more constraints and control. And how do architects know there is a gap? It should be an hypothesis-driven process, that needs to establish that there is a valuable, usable, feasible future state.

How: Execution

Another possible objection against agile methods is that the processes in EA, and in the enterprise generally, are simply not operating with a time window of the typical sprint length of three weeks. This, of course, is true. But it is at closer inspection not a counter-argument against the application of agile principles to EA—just the opposite. The long process cycles add to EA's lack of transparency and promote a silo mentality. Agile techniques can help here. [Bente2012]

— Bente et al, Collaborative Enterprise Architecture

As an *executing capability*, architecture operates in various ways:

- Planning and analysis
- Governance and approvals
- Collaboration and guidance

Ideally, planning and analysis occurs "upstream" of the creation of a product team. In that guise, architecture functions as a sort of zoning or planning authority—"architecture" is not a process or organization directly experienced by the product team. In this ideal, there is no conflict with product teams because once the team is formed, the architect's job is done. However, this assumes that all the planning associated with launching a new product or capability was done correctly, and this itself is a kind of waterfall assumption. Some form of feedback and coordination is required in *multi-product environments*.



Figure 38. What is your Cost of Delay? [40: Image credit <https://www.flickr.com/photos/julianlim/4598412264>, downloaded 2016-10-25, commercial use permitted]

It is in the "governance and approval" kind of activity that conflict is most likely to emerge. Cadence and synchronization (e.g. processes) with the potential to block teams from pursuing their mission should be very carefully considered. If there is a process or a queue of architecture approvals, it at least should be operated on Cost of Delay of the work it's blocking. And more generally, across the organization, the process should be tested against an economic model such as establishing a nominal or [portfolio-level Cost of Delay](#). Like other processes, architecture itself can be assessed against such a baseline.

Queued approvals are only one way of solving issues. A rich and under-utilized approach is using internal market-type mechanisms, where overall rules are set and teams make autonomous decisions based on those rules. Don Reinertsen, in the *Principles of Product Development Flow*, discusses how Boeing implemented distributed decision-making through setting tradeoff rules for cost and weight. Rather than constantly routing design approvals through a single control point, Boeing instead set the principle that project managers could "purchase" design changes up to \$300 per unit, to save a pound of weight. As Reinertsen notes,

The intrinsic elegance of this approach is that the superiors didn't actually give up control over the decision. Instead, they recognized that they could still control the decision without participating in it. They simply had to control the economic logic of the decision. [\[Reinertsen2009\]](#), p 42.

One particular work product that architects often are concerned with is documentation. The desire for useful documentation, as mentioned above, reflects architecture's goals of curating a common ground for collaboration. As Bente notes, In an agile project, explicit care must be taken to ensure proper documentation—for example, by stating it as part of the condition of satisfaction of a user story or in the definition of done" [\[Bente2012\]](#) p 170

Architecture Kata

...standardization on a particular toolchain or technology stack is neither necessary nor sufficient for achieving enterprise architecture goals such as enabling teams to respond rapidly to changing requirements, creating high-performance systems at scale, or reducing the risk of intrusion or data theft. Just like we drive product and process innovation through the Improvement Kata, we can drive architectural alignment through it too.

Architectural goals—for example, desired performance, availability, and security—should be approached by iteratively specifying target conditions at the program level. Following the Principle of Mission, set out a clear vision of the goals of your enterprise architecture without specifying how the goals are to be achieved, and create a context in which teams can determine how to achieve them through experimentation and collaboration. [\[Humble2013\]](#), chap. 10.

— Humble/Molesky, Lean Enterprise

[Toyota Kata](#) was discussed in Chapter 7. In *Lean Enterprise*, Jez Humble and Joanne Molesky argue

that it can provide a useful framework for architecture objectives. Toyota Kata emphasizes end-state goals ("target conditions") and calls for hands-on investigation and response by participating workers, not consultants or distant executives. Architecture can benefit by understanding "gaps" in the sense of Toyota's target conditions, and then supporting teams in their collaborative efforts to understand and achieve the desired state. The [architectural impact](#) model can assist in thinking through suitable target conditions for architecture:

- top-line impact through re-use (lowering Cost of Delay)
- bottom-line impact through portfolio rationalization
- risk impact through minimizing attack surface and re-use of known good patterns and platforms



Figure 39. Australian strangler vine surrounding tree [41: Image credit <https://www.flickr.com/photos/cynren/16011788979>, downloaded 2016-10-23, commercial use permitted]

Keeping the target operating condition specific is preferable. When architecture scopes problems too broadly, the temptation is to undertake [large and risky transformation programs](#). As an alternative, Humble and Molesky suggest the "strangler pattern," proposed by Martin Fowler in 2004 [[Fowler2004a](#)]. This pattern uses as a metaphor Australian "strangler" vines that grow around trees until the original tree dies, at which point the strangler vine is now itself a sturdy, rooted structure (see picture).

To use the strangler pattern is not to replace the system all at once, but rather to do so incrementally, replacing one feature at a time. This may seem more expensive, as it means that both the old and new systems are running (and cost savings through sunsetting the old system will be delayed.) But the risk of replacing complex systems is serious, and needs to be considered along with any hoped-for cost savings through replacement. Humble and Molesky suggest:

- Start by delivering new functionality—at least at first
- Do not attempt to port existing functionality unless it is to support a business process change
- Deliver something fast
- Design for testability and deployability

The strangler pattern is proven in practice. Paul Hammant provides a large number of strangler pattern case studies, including:

- Airline booking application
- Energy trading application
- Rail booking application

and others [\[Hammant2013\]](#).

Of course, there are other ways architecture might add value beyond system replacement, in which case the strangler pattern may not be relevant. In particular, architects may be called on to closely collaborate with product teams when certain kinds of issues emerge. This is not a governance or control interaction; it is instead architecture as a form of shared consulting "bench." Not every product team needs a full time architect, the reasoning goes, so architects can be assigned to them on a temporary basis, e.g. for one or a few sprints, perhaps of the technical "spike" (discovery/validation/experimentation) variety.

In order to successfully meet this role, the architect needs to have hands-on technical ability. Many Agile authors are dismissive of "ivory-tower" architects who do not do "hands on" work, and in fact if an architect is going to sit with a technical team as a solutions advisor they clearly need the technical skills to do so. On the other hand, not all architects operate at the solutions level, nor are the problems they face necessarily programming problems. See sidebar, "The challenge of the *hands-on* architect."

The challenge of the "hands-on architect"

Architect is a broad category as we have seen. It includes individuals who are talented at single-product designs, as well as those tasked with managing the overall interactions between hundreds of systems.

It is well and good for architects to maintain some technical facility, but in the case of true, portfolio-level enterprise architects, how to do so may not be obvious. What if one's portfolio includes multiple platforms and languages? It is simply not possible to be hands-on in all of them. Some of the most challenging systems may be a complex mix of commercial product and customization, e.g. ERP or core banking systems. Choosing to be "hands on" may not even be welcomed by a given team, who may see it as meddlesome. And other teams may feel the architect is "playing favorites" in their choice of platform to be "hands-on" with.

Clearly, if the organization is running primarily on (for example) Node.js, having strong Javascript skills is important for the architect. But in more heterogeneous environments the architect may find strong data management skills to be more useful, as often interfaces between systems become their primary concern.

Another form of being "hands on" is maintaining good systems administration skills, so that the architect can easily experiment with new technologies. This is different from being adept in a given programming language. One recent positive trend is lightweight virtualization. In years past, experimenting with new products was difficult on two fronts:

- First, one had to obtain high performance computing resources capable of running demanding software. Sometimes these resources needed unusual operating systems (e.g. "in order to try our software, you have to run HP-UX version 11"—not a capability most architects had in their back pocket.)
- Second, one had to obtain demonstration version of software from vendors, who would usually start a sales cycle if you asked for it.

Times have changed. Demonstration versions of software are increasingly available with little overhead or risk of triggering unwanted sales calls. Platform requirements are less diverse. And lightweight virtualization (e.g. the combination of Vagrant and Virtualbox) now makes it possible for architects to be hands-on; modern laptops can run multiple VMs in cluster architectures. Significant experimentation can be carried out in working with systems of various characteristics. Being able to evaluate technologies in such a virtual lab setting is arguably even more useful than being a "coding architect." Product team developers do the programming; the architect should be more concerned with the suitability and feasibility of the integrated platform.

Evaluating architecture outcomes

Finally, how do we evaluate architecture outcomes? If an organization adopts an experimental, Toyota Kata approach, it may find that architecture experiments run on long time horizons. Maintaining an organizational focus on value may be challenging, as the experiments don't yield results quickly. Curating a common ground of understanding may sound like a fine ideal, but how do we measure it?

First, the concept of Net Promoter Score is relevant for any service organization, internal or external. Its single question "Based on your experience, on a scale of 1-10 would you recommend this product or service to a friend?" efficiently encapsulates value in a single, easy to respond to query.

As digital pipelines become more automated, it may be possible to evaluate their "[digital exhaust](#)" to evaluate the impact of architecture services:

- are architecture standards evident in the source and package managers?
- are platform recommendations encountering performance or capacity challenges?

In a world of increasing connectivity and automation, there is no reason for architects in the organization to lack visibility into the consequences of their recommendations. Ultimately, if the cost of operating the coordination mechanism that is architecture exceeds the value it provides, then continuing to operate it is irrational.

Portfolio management

...every product and every activity of a business begins to obsolesce as soon as it is started. Every product, every operation, and every activity in a business should be put on trial for its life every two or three years.

Each should be considered the way we consider a proposal to go into a new product, a new operation, or activity – complete with budget, capital appropriation request, and so on.

One question should be asked of each: "if we were not in this already, would we now go into it?" And if the answer is "no," the next question should be: "How do we get out and how fast?" [\[Drucker1963\]](#)

— Peter F. Drucker

The aggregate digital investments of any large enterprise are massive. Whether capital or expense, whether internally hosted or externally sourced, the IT portfolio consumes tremendous amounts of time, money, and expertise.

In this chapter, we have discussed architecture in terms of [catalogs](#), [diagrams](#), and [matrices](#), sometimes stored in architecture repositories. But architecture repositories are in general not [analytic](#) tools. Nor is architecture an analytic discipline (it is not usually strongly quantitative). Architecture too often relies heavily on interviews and expert opinions, approaches that are sometimes critiqued as relying on the "[Highest Paid Person's Opinion](#) ." Portfolio management can be a means to bringing in a more quantitative approach.

First, the definition of portfolio. [\[Betz2011\]](#) defines it as "things of consequence managed over long time horizon." The concept of [TOGAF catalog](#) is a good place to start. Frequently, portfolio

management starts at the application level; the application portfolio can be seen as a sort of alternate "chart of accounts" by which digital investments can be grouped.

Portfolios are intended to provide a consistent basis for comparison and understanding. **Items in the portfolio should be comparable.** They may rely on both objective and subjective data points:

- Objective data
 - Business revenue/value
 - Cost
 - Risks
 - Staffing
 - Service levels, changes, incidents
 - Product obsolescence (quantifiable technical debt)
- Subjective data
 - Usability
 - Customer satisfaction (net promoter score)
 - Engineering assessments (subjective perceptions of technical debt)

Digital investments and costs typically include some or all of the following:

- Hardware investment, depreciation or leasing
- Software licensing (typically 15%-20% annually of initial acquisition, required for vendor support).
- Floor space – i.e., real estate charges
- Facilities infrastructure: power, high volume air conditioning (HVAC), raised floor, racks, etc.
- Network connectivity and related infrastructure (e.g. directory services software, security perimeters, and the like)
- Operational software infrastructure: monitoring systems, batch schedulers, backup systems, and so on, all with their own associated costs.
- Operations and support staff. Staffing typically can come in various flavors:
 - Data center operations monitors
 - Help desk operators
 - Application specialists
 - Senior engineers
 - Senior IT executives and customer relationship managers
 - Business-side lead users and process and information specialists
 - Vendor relationship owners and contract managers

Application value analysis

Application portfolio management may concern itself with any or all of the following:

- Application functions (examined for redundancy)
- Application total cost of ownership
- Application complexity
- Fitness and currency of underlying technical products
- Application service performance
- Application customer feedback

A four-box is often used for application value analysis:

	Low Technical Fitness	High Technical Fitness
High Business Value	Re-engineer or replatform. Consider outsourcing carefully.	Invest as needed to exploit value.
Low Business Value	Retire if possible or outsource	Improve understanding of customer requirements Retire service if no longer serving a purpose but reclaim/re-use platform, capabilities & assets

Application rationalization

What does it mean to rationalize"? There are three steps:

1. Take an inventory of the items to be rationalized.
2. Categorize them to identify redundancy.
3. Consolidate redundancy

This implies some basis for classification, so that the investments can be compared. This is where industry taxonomies, such as found in [vertical standards](#), may help. You may call your application "PeopleSoft HR," "Workday," or even an internal brand like "HR2020," but a reference taxonomy categorizes it as simply a "Human Resource Management System."

[Data integrations](#) may also be a way to identify redundancy. When two systems start exchanging more and more data, a useful question is whether there is still a need for both or if a more economical, integrated solution is possible. Just beware of the [risk of overly ambitious consolidation efforts](#).

Through analysis and understanding of redundancy, business and technical value, applications (and related concepts such as services and capabilities) can be managed as a coherent investment strategy. For further information, see the books referenced at the end of this chapter.

Architecture standards

This section briefly describes a number of key architecture standards for the student's reference.

Business architecture

NOTE We previously discussed the Business Analysis Body of Knowledge. Although they have similar names, BABOK and BIZBOK are distinct in terms of content and their sponsoring organizations.

- The *Business Analysis Body of Knowledge* (BABOK), from the International Institute of Business Analysis [[IIBA2015](#)], focuses on business analysis: "the practice of enabling change in an enterprise by defining needs and recommending solutions that deliver value to stakeholders."
- The Business Architecture Body of Knowledge (BIZBOK) is a broader, more diverse set of guidance from the BA Guild [[BAGuild2016](#)]. It defines business architecture as "a blueprint of the enterprise that provides a common understanding of the organization and is used to align strategic objectives and tactical demands."

TOGAF

The Open Group Architecture Framework is a general purpose framework and method, currently in Version 9 [[Open2009](#)]. TOGAF dates from 1995 and is based on even earlier work from the US Department of Defense, dating back to 1986.

Archimate

Archimate is an architectural notation specification, now owned by The Open Group.

Industry verticals

Industry sectors often develop architectural representations of their concerns. Examples include:

- Association for Retail Standards (National Retail Federation)
- Banking Information Architecture Network (BIAN)
- The Enhanced Telecom Operating Model and related work of the Tele-Management Forum
- The ACORD architecture for insurance
- The Exploration, Mining, Metals and Minerals (EMMM™) Forum of The Open Group

Governmental frameworks

Governmental architecture frameworks are often used to standardize architecture, for example:

- The U.S. Federal Enterprise Architecture Framework (FEAF)
- The U.S. Department of Defense Architecture Framework (DODAF)

- The U.K. Ministry of Defense Architecture Framework (MODAF)

This section to be expanded in future editions.

Topics

Architecture, queues, and Cost of Delay

The following is an exploration of using Cost of Delay and impact mapping worked TLM example

TLM should route via corp governance & perf mgmt, align w/vendor mgmt... find sources on how these processes work in non IT domains... (have already discussed in book)

===== digital strategy

===== complex systems

- Describe Snowden's Cynefin and its major elements
- complexity (w/r/t Burgess)
- stability, state, etc (Burgess)
- complex system failures (Allspaw sidebar?)
- antifragility
- Cynefin
- self-steering, autopoeisis
- post-industrial

is EA using the right tools? systems theory, semantics vs dynamics.

===== EA as systems thinking
hard to argue with but evidence is limited

Sidebar: Introducing Burgess.

Sidebar: Introducing Snowden & Taleb.

- Postscript: Thought experiment on complete mainstreaming of IT

arch formalisms weak on dynamics - semantics vs dynamics problem semantics gravitate to first order predicate logic, binary thinking continuous math... required... how to measure anything. systems thinking.

===== EA and design thinking

Design thinking is encroaching on architecture. Enterprise design?
the sheer absurdity of undertaking something so ambitious, on the one hand, and, foolhardy, on the other, as any kind of intentional design of so complex and poorly (and even mis-) understood an entity as an organization in a context of rapid change._<<Malan2010>>

===== metamodels

===== tech platform as compensation

Note that we compensate developers with money AND experience. Developer pressure to use a new technology should be seen in part as a demand for increased compensation. developers seeking experience, it is in part how they are compensated. this leads to conflicts over choice of tools. (good blog as well). rational framework for deciding. might go to chapter 8, resource mgmt. intersection between that & tech lifecycle is key.

===== SAFE and architecture

Review SAFE on architecture, also Reinertsen

===== Sol Arch for EA

===== misc

"Architecture of software is a collection of design decisions that are expensive to change." <<Ran2001>>

[quote, Ruth Malan]

For enduring competitive advantage, the organization needs to seek leverage across its products and/or services and to create strategic capabilities that are hard to imitate because they rely on a mix of process, technology, skills, resources, facilities, culture, and even history <<Malan2005>>

<https://structurizr.com/help/about>

All arch is design not all design is arch (Booch)

===== NOTES

Cathedral / bazaar

Bente's four perspectives & eight caricatures

[quote, Bente et al]

With regard to the nature of complex systems, it is not advisable to search for the "optimal enterprise architecture." Optimality is always relative to an environment, and the frame conditions are likely to change. Hence it is better to accept certain ugly spots of the IT landscape and strengthen the viability of the whole, which means developing and using capabilities such as being able to learn, solve problems, adapt, or absorb disruptions.

critical importance of not confusing solutions with EA. Scale matters.

from bente: Anti-patterns in EA and architecture in general can be found, for instance, in Kruchten (2008), Mar and Spacey (2011), Ambler, Nalbone. and Vizdos (2005), and Coplien and Harrison (2005).

<https://philippe.kruchten.com/2011/02/13/the-elephants-in-the-agile-room/>

<http://www.mckinsey.com/business-functions/organization/our-insights/to-centralize-or-not-to-centralize?platform=hootsuite>

review "a day in the life of an enterprise architect & Integration of EA & App Portfolio" walker, Comparison of top 4 EA methods, Sessions

Conclusion

Architecture is a challenging topic for both the student and the practitioner. Its value is too often unclear, but when it is eliminated, it tends to re-emerge in response to operational learnings. "If we'd just had some architecture, X would not have happened" is too often heard.

As humans we think in abstractions and architecture is an expression of this. The architect skilled in doing so, in sense-making across the landscape and providing guidance on key decisions, identifying opportunities, and taking care for stewardship of the longer horizon will remain a valuable asset to the digital organization as it scales up into the largest problems.

Discussion questions

- Should digital professions continue to use the term "architect"?
- Debate the pros and cons of having an architecture organization approve designs or technology platform choices. Be prepared to argue either side.
- Debate the pros and cons of visual modeling
- Do you agree that architecture is a form of staff function? Why or why not?
- Read the articles by Bloomberg below and discuss whether there is any future in architecture.
 - [Is Enterprise Architecture Completely Broken?](#)
 - [Agile Enterprise Architecture Finally Crosses the Chasm](#)

Research & practice

- Build a virtual machine cluster and install or develop software for it, ideally including a database. Download the free [Archi](#) tool for Archimate and draw a representation of the software or data architecture.
- Draw the same idea in two different ways in Archimate.
- Draw two or three catalogs and associated matrices and diagrams. Make sure all are consistent (e.g. lines correspond to matrix intersection data)
- Compare and contrast any two [industry vertical architectures](#).
- Compare and contrast business model versus operating model, and in the same analysis consider the distinction between enterprise architecture and business architecture. Can enterprise architecture be equated with the operating model? If business architecture is a subset of enterprise architecture, does that mean it also is primarily concerned with the operating model? Or is business architecture also concerned with the business model?

Further reading

Books

Stefan Bente, [Collaborative Enterprise Architecture: Enriching EA with Lean, Agile, and Enterprise 2.0 Practices](#)

Chris Potts, [RecrEAtion: Realizing the Extraordinary Contribution of Your Enterprise Architects](#)

Jeanne Ross, [Enterprise Architecture As Strategy: Creating a Foundation for Business Execution](#)

Steven Spewak, [Enterprise Architecture Planning: Developing a Blueprint for Data, Applications, and Technology](#)

William Ulrich, [Business Architecture: The Art and Practice of Business Transformation](#)

Articles

- [Agile, TOGAF and Enterprise Architecture: Will They Blend?](#)
- [Is Enterprise Architecture Completely Broken?](#)
- [Agile Enterprise Architecture Finally Crosses the Chasm](#)
- [Let's Go to the Whiteboard: How and Why Software Developers Use Drawings](#)
- [Booch: Why Don't Developers Draw Diagrams?](#)
- [Malan: Why Don't Developers Draw Diagrams?](#)
- [Malan: Reworking some of the ideas: Why Don't Developers Draw Diagrams?](#)
- [Booch: Draw Me a Picture](#)
- [Malan: When Are Diagrams Useful?](#)

Professional

- [Business Architecture Guild](#)
- [Association of Enterprise Architects](#)
- [Iasa Global, a non-profit association for ALL IT Architects](#)

Section and book conclusion

At this writing, there is much debate on how to unify the value and insights of team-based Agile development, with the coordination and governance needs of the large enterprise. Thinking in terms of:

- Governance
- Information
- Architecture

is recommended as the enterprise digital experience requires their formalization. But it is critical to not lose track of the product vision and value you started with at the outset of our emergence journey. Without fast feedback enabling responsive digital services for the customer, none of the rest of it matters.

Congratulations, and thank you for persisting through this work.

Appendices & back matter

placeholder for appendices

- Frameworks overview
- Architectural views (??)
- Towards a theory of IT management

The major frameworks

Concept introduced in Section III, reprinted summary here

Towards a theory of IT management

As we look forward, and consider the scale at which we are now operating, the challenges only seem to increase. Increasingly, we need to look to engineering and operations practices from other fields, and to math and science for useful theoretical tools.

Risk management, complex systems failures, how to safely manage combined human/technical systems - this is all well studied in domains like aerospace. We can't understand "change" management without better definitions of change and stability, concepts well studied in the quantitative sciences. No matter the safeguards, complex systems are prone to fail - but can we conceive of systems that can increase their stability through learning from their failures?

Finally, what do we even mean by complexity? What is the relationship between the merely complicated, the truly complex, and the absolutely chaotic? What do we need to understand about these various domains, to continue our journey of digital transformation?

Folk models = "sr. clinician's opinions" - Nancy Raymond

the problem of frameworks - folk applications of semantics

traditions of canon law (see chapter 11)

Optimizing across multiple constraints

Limits of iterative approaches ... missing higher order modes ...

- Reinertsen's critique of CC

What we can and cannot learn from simulations (AB Murray)

References

NOTE

All web references were archived into the author's files on the indicated access date. Please contact the author if a link is broken.

- [Accounting2016]: Accounting Coach. 2016. "What Is Cost Accounting?" <http://www.accountingcoach.com/blog/what-is-cost-accounting> (May 7, 2016).
- [Adzic2012]: Adzic, Gojko. 2012. Impact Mapping: Making a Big Impact with Software Products and Projects. Gojko Adzic.
- [Akera2007]: Akera, A., "Edmund Berkeley and the Origins of the ACM." Communications of the ACM, 2007. 50(5): p. 31-35.
- [Alliance2001]: Agile Alliance. (2001), "Agile Manifesto and Principles", <http://agilemanifesto.org/principles.html>
- [Alliance2015]: Agile Alliance. 2015. "Team Definition." Glossary. <https://www.agilealliance.org/glossary/team/>. Accessed 2016-09-21.
- [Allspaw2009]: Allspaw, J. & Hammond, P. (2009), "10 deploys per day: Dev & ops cooperation at Flickr", O'Reilly Publications, <http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr>
- [Ambler2005]: Ambler, S. (2005) "Agile Outsourcing." *Dr. Dobb's Journal*. Retrieved from <http://www.drdobbs.com/architecture-and-design/agile-outsourcing/184415344>. Date Accessed: 3/21/2016.
- [Ambler2006]: Ambler, Scott W, and Pramod J Sadalage. 2006. Refactoring Databases : Evolutionary Database Design. Book. Harlow, U.K.: Addison-Wesley. Table of contents <http://www.loc.gov/catdir/toc/ecip063/2005031959.html>.
- [Ambler2012]: Ambler, Scott W, and Mark Lines. 2012. Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press.
- [Anderson2010]: Anderson, D. J. (2010), *Kanban: Successful Evolutionary Change for your Technology Business*, Blue Hole Press, Sequim, WA
- [Arbogast2012]: Arbogast, T. L., Craig; Vodde, Bas (2012) "Agile Contracts Primer." Retrieved from http://www.agilecontracts.com/agile_contracts_primer.pdf. Date Accessed: March 21, 2016.
- [Ashmore2014]: Ashmore, S. and K. Runyan, 2014, *Introduction to agile methods*. xix, 307 pages.
- [Bacik2011]: Bacik, S. (2011). Compliance, audit, risk, security – what's the difference and why do we need it? Retrieved July 6, 2016, from <http://www.enernex.com/wp-content/uploads/2011/11/Risk-Compliance-Audit-webinar-11.2011.pdf>
- [BAGuild2016]: Business Architecture Guild. 2016. A Guide to the Business Architecture Body of Knowledge (BIZBOK Guide). Business Architecture Guild.
- [Bank2016]: Bank, C., & Cao, J. (n.d.). The Guide to Usability Testing. uxpin.com. Retrieved from <https://www.uxpin.com/studio/ebooks/guide-to-usability-testing/> (free ebook, accessed 2016-09-22)
- [Beck2000]: Beck, K., 2000 *extreme programming eXplained : embrace change*, Reading, MA: Addison-Wesley. xxi, 190 p.
- [Bell2010]: Bell, S. C. & Orzen, M. A. (2010), *Lean IT*, CRC Press, Boca Raton, Florida
- [Bell2013]: Bell, Steve et al. 2013. *Run Grow Transform : Integrating Business and Lean IT*. Boca Raton, FL: CRC Press.
- [Bente2012]: Bente, Stefan, Uwe Bombosch, and Shailendra Langade. 2012. Collaborative Enterprise Architecture: Enriching EA with Lean, Agile, and Enterprise 2.0 Practices. Waltham,

MA: Morgan Kaufman - Elsevier.

- [Bernard2012]: Bernard, Scott. 2012. *An Introduction to Enterprise Architecture*. AuthorHouse.
- [Betz2011]: Betz, C. (2011), "Release management integration pattern - seeking devops comments", <http://www.lean4it.com/2011/01/release-management-integration-pattern-seeking-devops-comments.html>
- [Betz2011a]: Betz, C. T. (2011), *Architecture and Patterns for IT: Service and Portfolio Management and Governance (Making Shoes for the Cobbler's Children)*, 2nd Edition, Elsevier/Morgan Kaufman, Amsterdam
- [Betz2011b]: Betz, C. (2011). "ITIL®, COBIT®, and CMMI®: Ongoing Confusion of Process and Function." BPTrends. Retrieved from <http://www.bptrends.com/publicationfiles/10-04-2011-ART-Ongoing Confusion of Process and Function-Betz-Final.pdf>
- [Betz2015]: Betz, C.T. (2015), "Calavera project", Github, <https://github.com/CharlesTBetz/Calavera>
- [Binder1997]: Binder, R.V. 1997. "Can a Manufacturing Quality Model Work for Software?" IEEE Software 14 (5). IEEE: 101–2, 105. doi:10.1109/52.605937.
- [Blank2013]: Blank, Steve. 2013. *The Four Steps to the Epiphany: Successful Strategies for Products That Win*. 2nd ed. Steve Blank, publisher.
- [Bloomberg2014]: Bloomberg, Jason. 2014. "Agile Enterprise Architecture Finally Crosses the Chasm." Forbes. July. <http://www.forbes.com/sites/jasonbloomberg/2014/07/23/agile-enterprise-architecture-finally-crosses-the-chasm/print/>. Accessed 2015-11-12.
- [BPTrends2013]: BPTrends. 2013. "Scope Diagram (or Input-Guides-Outputs-Enablers – IGOE Diagram)." BPTrends. <http://www.bptrends.com/resources/glossary/scope-diagram-or-input-guides-outputs-enablers-igoe-diagram/>.
- [Brooks1995]: Brooks, Frederick P. 1995. *The Mythical Man-Month: Essays on Software Engineering*. 25th Anniversary edition. Reading, Mass.: Addison-Wesley Pub. Co.
- [Bossavit2015]: Bossavit, L., 2015. *The Leprechauns of Software Engineering: How folklore turns into fact and what to do about it* 2015. Available from: <https://leanpub.com/leprechauns>.
- [Bouwman]: Bouwman, J.-J. & Heistek, M. O, "ITIL and DevOps at war in the enterprise," https://www.youtube.com/watch?v=_dDsdbkSgOc, DevOpsDays.
- [Brooks1975]: Brooks, F. P. (1975). *The mythical man-month: essays on software engineering*. Reading, Mass., Addison-Wesley Pub. Co.
- [Brustein2013]: Brustein, Joshua. 2013. "Microsoft Kills Its Hated Stack Rankings. Does Anyone Do Employee Reviews Right?" Bloomberg Business Week. <http://www.bloomberg.com/news/articles/2013-11-13/microsoft-kills-its-hated-stack-rankings-dot-does-anyone-do-employee-reviews-right> (March 5, 2016).
- [Buchholz1962]: Buchholz, Werner. 1962. *Planning a Computer System: Project Stretch*. New York: McGraw-Hill Book Company, Inc.
- [Buckingham2015]: Buckingham, Marcus, and Ashley Goodall. 2015. "Reinventing Performance Management." Harvard Business Review 93(4): 40–50. <https://hbr.org/2015/04/reinventing-performance-management>.
- [Burlton2001]: Burlton, Roger. 2001. *Business Process Management: Profiting from Process*.

Indianapolis, Indiana: SAMS.

- [Burgess2015]: Burgess, Mark. 2015. *Thinking in Promises*. Sebastopol, CA: O'Reilly Media.
- [Burrows2006]: Burrows, M. G. I. (2006). "The Chubby lock service for loosely-coupled distributed systems." 7th symposium on Operating systems design and implementation (OSDI '06), USENIX Association Berkeley, CA, USA
- [Burrows2014]: Burrows, Mike. (2014) *Kanban from the Inside: Understand the Kanban Method, connect it to what you already know, introduce it with impact*. Sequim, WA, Blue Hole Press.
- [Buschmann1996]: Buschmann, F. (1996), *Pattern-oriented software architecture : a system of patterns*, Wiley, Chichester ; New York
- [Butler2013]: Butler, Brandon. 2013. "Free Cloud Storage Service MegaCloud Goes Dark." Network World. doi:<http://www.networkworld.com/article/2171450/cloud-computing/free-cloud-storage-service-megacloud-goes-dark.html>. Accessed 2016-07-01.
- [Butler2014]: Butler, Brandon. 2014. "Cloud's Worst-Case Scenario: What to Do If Your Provider Goes Belly up." Network World. <http://www.networkworld.com/article/2173255/cloud-computing/cloud-s-worst-case-scenario-what-to-do-if-your-provider-goes-belly-up.html>. Accessed 2016-07-01.
- [Cadbury1992]: Committee on the Financial Aspects of Corporate Governance. 1992. "Report of the Committee on the Financial Aspects of Corporate Governance (aka Cadbury Report)." London, Gee & Co. Ltd.
- [Cagan2008]: Cagan, Marty. 2008. Inspired: How to Create Products Customers Love. SVPG Press. <http://www.amazon.com/Inspired-Create-Products-Customers-Love/dp/0981690408>.
- [Card1999]: Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). Readings in Information Visualization: Using Vision to Think. San Diego: Academic Press.
- [Carr2003]: Carr, N. (2003). IT Doesn't Matter. Harvard Business Review, 5–12.
- [Carroll2013]: Carroll, I. (2013), "Various", <http://itopskanban.wordpress.com/before/>
- [Castaldo2016]: Castaldo, Joe. 2016. "The Last Days of Target: The Untold Tale of Target Canada's Difficult Birth, Tough Life and Brutal Death." Canadian Business. <http://www.canadianbusiness.com/the-last-days-of-target-canada/>. Accessed 2016-08-30.
- [Chacon2009]: Chacon, S. & Straub, B. (2009). Pro Git. Berkeley, CA. New York, Apress.
- [Cherubini2007]: Cherubini, M., Venolia, G., Deline, R., & Ko, A. J. (2007). Let ' s Go to the Whiteboard: How and Why Software Developers Use Drawings. CHI 2007 Proceedings, 557–566. <http://doi.org/10.1145/1240624.1240714>. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/p557-cherubini.pdf>, accessed 2016-10-17.
- [Chisholm2001]: Chisholm, M. (2001). Managing Reference Data in Enterprise Databases: Binding Corporate Data to the Wider World. San Diego: Academic Press.
- [Christensen2006]: Christensen, Clayton, Scott Cook, and Taddy Hall. 2006. "What Customers Want from Your Products." Working Knowledge (Harvard Business School). <http://hbswk.hbs.edu/item/what-customers-want-from-your-products>. Accessed 2016-09-18.
- [Christensen2015]: Clayton Christensen Institute. 2015. "Jobs to Be Done." [Http://www.christenseninstitute.org/](http://www.christenseninstitute.org/). <http://www.christenseninstitute.org/key-concepts/jobs-to-be-done/>. Accessed 2016-09-18.

- [Clark2006]: Clark, Nicola. 2006. “The Airbus Saga: Crossed Wires and a Multibillion-Euro Delay.” New York Times, December 11. <http://www.nytimes.com/2006/12/11/business/worldbusiness/11iht-airbus.3860198.html>. Accessed 2016-10-11.
- [Coase1937]: Coase, R. (1937). The nature of the firm. *Economica*, 4, 386–405.
- [Cobb2015]: Cobb, C.G., (2015), *The Project MANAGER’S GUIDE TO MASTERING AGILE: Principles and Practices for an Adaptive Approach*, Hoboken, New Jersey: John Wiley & Sons.
- [Cockburn2007]: Cockburn, Alistair. 2007. Agile Software Development: The Cooperative Game. 2nd ed. Boston, MA: Pearson Education, Inc.
- [Cohn2010]: Cohn, M., *Succeeding with Agile: Software Development Using Scrum*, Addison-Wesley: Upper Saddle River, New Jersey.
- [Comella2016]: Comella-Dorda, Santiago, Lohiya, Swati, and Gerard Speksnijder. 2016. “An Operating Model for Company-Wide Agile Development.” McKinsey & Company. <http://www.mckinsey.com/Business-Functions/Business-Technology/Our-Insights/An-operating-model-for-company-wide-agile-development>.
- [Cooper2009]: Cooper, A., Reimann, R., & Cronin, D. (2009). About Face 3: The Essentials of Interaction Design. online. Retrieved from <http://www.amazon.com/About-Face-Essentials-Interaction-Design-ebook/dp/B008NC0XR2/>
- [COSO2013]: Committee of Sponsoring Organizations of the Treadway Commission. 2013. “Internal Control — Integrated Framework (Executive Summary).” http://www.coso.org/documents/990025P_Executive_Summary_final_may20_e.pdf.
- [Csikszentmihalyi1990]: Csikszentmihalyi, M. (1990). Flow : the psychology of optimal experience. New York, Harper & Row.
- [Cunningham1992]: Cunningham, Ward. 1992. “Experience Report: The WyCash Portfolio Management System.” OOPSLA ’92. <http://c2.com/doc/oopsla92.html>. Accessed 2016-10-6.
- [DAMA2009]: Data Management Association, The. 2009. The DAMA Guide to The Data Management Body of Knowledge (DAMA-DMBOK Guide). Bradley Beach, NJ: Technics Publications, LLC.
- [Davenport2007]: Davenport, Thomas H, and Jeanne G Harris. 2007. Competing on Analytics : The New Science of Winning. Boston, Mass.: Harvard Business School ; London : McGraw-Hill [distributor]. Table of contents only <http://www.loc.gov/catdir/toc/ecip073/2006035422.html>.
- [delaMaza2016]: de la Maza, Michael, and David Benz. 2016. Why Agile Works: The Values Behind the Results. C4Media - InfoQ.com. <http://www.infoq.com/resource/minibooks/why-agile-works>. Accessed 2016-10-11.
- [DeLuccia2008]: DeLuccia, James. 2008. *IT COMPLIANCE AND CONTROLS: Best Practices for Implementation*. Hoboken, N.J.: John Wiley & Sons, Inc.
- [DeLuccia2015]: DeLuccia, James, Jeff Gallimore, Gene Kim, and Byron Miller. 2015. “DevOps Audit Defense Toolkit.” <http://itrevolution.com/devops-and-auditors-the-devops-audit-defense-toolkit/>.
- [DeNicola216]: De Nicola, Antonio, and Michelle Missikoff. 2016. “A Lightweight Methodology for Rapid Ontology Engineering.” Communications of the ACM2 59 (3): 79–86.

- [DHS2006]: Department of Homeland Security. 2006. "Report No. 2006-03, The Use of Commercial Data." DHS Data Privacy and Integrity Advisory Committee.
- [Drucker1963]: Drucker, Peter F. 1963. "Managing for Business Effectiveness." Magazine Article. Harvard Business Review.
- [Drucker1993]: Drucker, Peter F. 1993. *Post-Capitalist Society*. 1st ed. New York, NY: HarperBusiness.
- [duPreez2015]: du Preez, Derek. 2015. "A CIO's Worst Nightmare: When Your Cloud Provider Goes Bankrupt." Diginomica. <http://diginomica.com/2015/01/06/cios-worst-nightmare-cloud-provider-goes-bankrupt/>. Accessed 2016-07-04.
- [Duvall2007]: Duvall, P. M.; Matyas, S. & Glover, A. (2007), *Continuous integration : improving software quality and reducing risk*, Addison-Wesley, Upper Saddle River, NJ
- [Edwards2012]: Edwards, D. (2012), "Integrating DevOps tools into a Service Delivery Platform", <http://dev2ops.org/2012/07/integrating-devops-tools-into-a-service-delivery-platform-video/>
- [Eisenhardt1989]: Eisenhardt, Kathleen M. 1989. "Agency Theory: An Assessment and Review." *Academy of Management Review* 14 (1): 57–74. <http://www.jstor.org/stable/258191>.
- [England2013]: England, Rob. 2013. *Plus! The Standard+Case Approach: See Service Response in a New Light*. Mana, New Zealand: Two Hills Ltd.
- [Evans2004]: Evans, Eric. 2004. Domain-Driven Design : Tackling Complexity in the Heart of Software. Book. Boston ; London: Addison-Wesley.
- [Fisher2016]: Fisher, T. (2016). Designing Our Way to a Better World. Minneapolis, MN: University of Minnesota Press.
- [Flahiff2016]: Flahiff, J. (2016). "How organizational agility will save and destroy your company." from <http://searchcio.techtarget.com/tip/How-organizational-agility-will-save-and-destroy-your-company>. Accessed March 19, 2016.
- [Forsgren2016]: Forsgren, Nicole, Gene Kim, Nigel Kersten, Jez Humble, and Alanna Brown. 2016. "2016 State of DevOps Report." Puppet Labs.
- [Fowler1997]: Fowler, M. (1997), *Analysis patterns : reusable object models*, Addison Wesley, Menlo Park, Calif.
- [Fowler2003]: Fowler, M. (2003), *Patterns of enterprise application architecture*, Addison-Wesley, Boston
- [Fowler2003a]: Fowler. 2003. "Who Needs an Architect?" IEEE Software, no. July/August. <http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>.
- [Fowler2004]: Fowler, Martin. 2004. "Is Design Dead?" Martinfowler.com. <http://martinfowler.com/articles/designDead.html>. Accessed 2016-10-10.
- [Fowler2004a]: Fowler, Martin. 2004. "Bliki: StranglerApplication." Accessed October 23. <http://martinfowler.com/bliki/StranglerApplication.html>.
- [Fowler2006]: Fowler, Martin. 2006. "Shu-Ha-Ri." Martin Fowler's Bliki. <http://martinfowler.com/bliki/ShuHaRi.html>.
- [Fowler2014]: Fowler, Martin. 2014. "BoundedContext." Martin Fowler's Bliki2. <http://martinfowler.com/bliki/BoundedContext.html>. Accessed 2016-09-01.

- [Furr2013]: Furr, N. A., Ahlstrom, Paul (2013). *Nail It then Scale It: The Entrepreneur's Guide to Creating and Managing Breakthrough Innovation*, NISI Publishing.
- [Gall2012]: Gall, John. 2012. The Systems Bible: The Beginner's Guide to Systems Large and Small. General Systemantics Pr/Liberty.
- [Gamma1995]: Gamma, E. (1995), *Design patterns : elements of reusable object-oriented software*, Addison-Wesley, Reading, Mass.
- [Gawande2010]: Gawande, Atul. 2010. *The Checklist Manifesto: How to Get Things Right*. New York, N.Y: Picador.
- [Glass1998]: Glass, R.L. (1998), *Software runaways*, Upper Saddle River, NJ: Prentice Hall PTR. xvi, 259.
- [Glen2003]: Glen, P. (2003). Leading Geeks: How to Manag and Lead People who Manage Technology. San Francisco, Jossey-Bass.
- [Goldratt1997]: Goldratt, E. M. (1997), *Critical chain*, North River, Great Barrington, Ma.
- [Goldratt2004]: Goldratt, E. M. & Cox, J. (2004), *The goal : a process of ongoing improvement*, North River Press, Great Barrington, MA
- [GoldrattUK2016]: Goldratt-UK (2016). "What is Critical Chain?". Retrieved 2/18/2016, from http://www.goldratt.co.uk/resources/critical_chain.
- [Goodwin2015]: Goodwin, B. (2015). How CIOs can raise their *IT clock speed* as pressure to innovate grows. ComputerWeekly.com. <http://www.computerweekly.com/feature/How-CIOs-can-ramp-up-their-IT-clock-speed-as-pressure-grows>.
- [Gothelf2013]: Jeff Gothelf, and Josh Seiden. 2013. Lean UX: Applying Lean Principles to Improve User Experience. Sebastopol, CA: O'Reilly Media, Inc.
- [Griffin2016]: Griffin, Michael. 2016. How To Write a Policy Manual. www.templatezone.com. Accessed 2016-07-03. <http://www.templatezone.com/download-free-ebook/office-policy-manual-reference-guide.pdf>.
- [Gruver2013]: Gruver, G., M. Young, and P. Fulghum, 2013, *A practical approach to large-scale Agile development : how HP transformed laserjet futuresmart firmware* xxiv, 183 pages.
- [Gulden tops2011]: Gulden tops, Erik. 2011. "Where Have All the Control Objectives Gone? They Have Picked Them Every One." ISACA Journal 4. <http://www.isaca.org/Journal/archives/2011/Volume-4/Documents/jpdf11v4-Where-Have-All.pdf>.
- [Hammant2013]: Hammant, Paul. 2013. "Legacy Application Strangulation : Case Studies." Paul Hammant's Blog. <http://paulhammant.com/2013/07/14/legacy-application-strangulation-case-studies/>.
- [Hammer1993]: Hammer, Michael, and James Champy. 1993. Reengineering the Corporation : A Manifesto for Business Revolution. Brealey Publishing.
- [Harmon2003]: Harmon, Paul. 2003. Business Process Change: A Manager's Guide to Improving, Redesigning, and Automating Processes. Amsterdam: Elsevier.
- [Harpring2010]: Harpring, Patricia. 2010. Introduction to Controlled Vocabularies: Terminology for Art, Architecture and Other Cultural Works. Los Angeles, CA: Getty Publications. http://www.getty.edu/research/publications/electronic_publications/intro_controlled_vocab/index.html.

- [Harris2013]: Harris, S. (2013). CISSP Exam Guide (6th ed.). New York: McGraw-Hill Education.
- [Hay1996]: Hay, D. C. (1996), *Data model patterns : conventions of thought*, Dorset House Pub., New York
- [Hay2006]: Hay, D. C. (2006), *Data model patterns : a metadata map*, Morgan Kaufmann ; Oxford : Elsevier Science [distributor], San Francisco, Calif.
- [Heller2016]: Heller, Martha. 2016. “GE’s Jim Fowler on the CIO Role in the Digital Industrial Economy.” CIO Magazine Online. <http://www.cio.com/article/3048805/leadership-management/ges-jim-fowler-on-the-cio-role-in-the-digital-industrial-economy.html>.
- [Hodges2016]: Hodges, Matt. n.d. “12 Steps to Creating Landing Pages That Convert.” Inside Intercom. Accessed 2016-09-18.
- [Hohpe2003]: Hohpe, G. & Woolf, B. (2003), *Enterprise integration patterns : designing, building, and deploying messaging solutions*, Addison-Wesley, Boston
- [Hope2001]: Hope, Jeremy, and Robin Fraser. 2001. Beyond Budgeting Questions and Answers. <http://bbrt.org/product/bbdt-qb-white-paper-october-2001/>.
- [Housman2015]: Housman, Michael, and Dylan Minor. 2015. “Toxic Workers.” http://www.hbs.edu/faculty/Publication Files/16-057_d45c0b4f-fa19-49de-8f1b-4b12fe054fea.pdf.
- [Hubbard2009]: Hubbard, Douglas W. 2009. *The Failure of Risk Management*. Hoboken, New Jersey: John Wiley & Sons, Inc.
- [Hubbard2010]: Hubbard, D. (2010), *How to Measure Anything: Finding the Value of Intangibles in Business*, Wiley, Boston
- [Humble2011]: Humble, J. & Farley, D. (2011), *Continuous delivery*, Addison-Wesley, Boston
- [Humble2013]: Humble, Jez, Joanne Molesky, and Barry O'Reilly. 2013. Lean Enterprise. Book. The Lean Series. First edit.
- [Humphrey1989]: Humphrey, Watts S. 1989. *Managing the Software Process*. Reading, Mass.: Addison-Wesley.
- [Huntzinger2007]: Huntzinger, James R. 2007. *Lean Cost Management: Accounting for Lean by Establishing Flow*. Fort Lauderdale, Fl.: J. Ross Publishing.
- [IAASB2013]: International Auditing and Assurance Standards Board (IAASB). 2013. “ISAE 3000 (Revised), Assurance Engagements Other than Audits or Reviews of Historical Financial Information.” <https://www.ifac.org>.
- [Inmon1992]: Inmon, William H. 1992. Building the Data Warehouse. Wiley.
- [IIBA2015]: International Institute of Business Analysis (IIBA). 2015. BABOK v3: A Guide to the Business Analysis Body of Knowledge. Toronto, Canada: International Intitute of Business Analysis.
- [Isaacs2002]: Isaacs, E., & Walendowski, A. (2002). Designing from both sides of the screen: How Designers and Engineers Can Collaborate to Build Cooperative Technology. Indianapolis, Indiana: New Riders Publishing.
- [ISACA2012]: ISACA. 2012. *COBIT 5: Enabling Processes*.
- [ISACA2012a]:ISACA. 2012. *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*. Rolling Meadows, IL: ISACA.

- [ISACA2012b]: ISACA. (2012). *COBIT 5 for Information Security*. Rolling Meadows, IL: ISACA.
- [ISACA2013]: ISACA. (2013). *COBIT 5 for Risk*. (ISACA, Ed.). Rolling Meadows, IL.
- [ISACA2013a]: ISACA. (2013). *COBIT 5 for Assurance*. Rolling Meadows, IL: ISACA.
- [ISACA2013b]: ISACA. (2013). *COBIT 5 Enabling Information*.
- [ISACA2014]: ISACA. 2014. ITAF: A Professional Practices Framework for IS Audit/ Assurance, 3rd Edition. Rolling Meadows, IL: ISACA.
- [ISO2008]: ISO/IEC. 2008. "ISO/IEC 38500 - Corporate Governance of Information Technology."
- [ISO2009]: ISO/IEC. 2009. "ISO 31000:2009 - Risk Management."
- [ISO2011]: ISO/IEC/IEEE. 2011. "ISO/IEC/IEEE 42010:2011 - Systems and Software Engineering—Architecture Description." Vol. 2011. doi:10.1109/IEEESTD.2011.6129467.
- [Izrailevsky2011]: Izrailevsky, Y., & Tseitlin, A. (2011). The Netflix Simian Army. Retrieved May 4, 2016, from <http://techblog.netflix.com/2011/07/netflix-simian-army.html>
- [Kan2003]: Kan, Stephen H. 1995. *Metrics and Models in Software Quality Engineering*. Second Edition. Reading, Mass.: Addison-Wesley.
- [Kaner1999]: Kaner, C., Falk, J. L., & Nguyen, H. Q. (1999). Testing computer software (2nd ed.). New York: Wiley.
- [Kaplan1992]: Kaplan, Robert S., and David P. Norton. 1992. "The Balanced Scorecard - Measure That Drive Performance." Harvard Business Review, no. January-February: 71–79. doi:00178012.
- [Keefer2006]: Keefer, G. "The CMMI Considered Harmful For Quality Improvement And Supplier Selection." 2006. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.4292&rep=rep1&type=pdf>
- [Kennaley2010]: Kennaley, M., 2010. *SDLC 3.0: Beyond a Tacit Understanding of Agile: Towards the Next Generation of Software Engineering* Fourth Medium Consulting.
- [KARE2015]: KARE 11 Staff. 2015. "Target Cuts 275 Positions, Most in Technology." <http://www.kare11.com/story/news/2015/09/01/target-cuts-275-jobs-most—technology/71512016/>.
- [Kiley2001]: Kiley, Kevin. 2001. "The Grand Quartier-General Imperial and the Corps d'Armée: Developments in the Military Art, 1795-1815." Military Subjects: Organization, Strategy & Tactics. http://www.napoleon-series.org/military/organization/c_staff1.html. Accessed 2016-10-04.
- [Kim2013]: Kim, G.; Behr, K. & Spafford, G. (2013), *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, IT Revolution Press
- [Klein2005]: Klein, Gary, Paul J. Feltovich, and David D. Woods. 2005. "Common Ground and Coordination in Joint Activity." In *Organizational Simulation*. Hoboken, New Jersey: John Wiley & Sons, Inc.
- [Knez2002]: Knez, Mark, and Duncan Simester. 2002. "Making Across-the-Board Incentives Work." Harvard Business Review (Feb 2002).
- [Kniberg2011]: Kniberg, H.; Beck, K. & Keppler, K. (2011), *Lean from the trenches : managing large-scale projects with Kanban*, Pragmatic Bookshelf, Dallas, Tex.
- [Kniberg2013]: Kniberg, Henrik. 2013. "Culture Over Process." Youtube. <https://www.youtube.com/watch?v=Rb0O0Lgs9zU>.

- [Kohavi2009]: Kohavi, Ronny, Thomas Crook, and Roger Longbotham. 2009. “Online Experimentation At Microsoft.” Online. <http://www.experimentplatform.com/Documents/ExPThinkWeek2009Public.pdf>. Accessed 2016-09-22.
- [Koskela2002]: Koskela, L.H., Gregory The underlying theory of project management is obsolete. 2002. http://www.researchgate.net/publication/3229647_The_Underlying_Theory_of_Project_Management_Is_Obsolete
- [Krafcik1988]: Krafcik, J. (1988). "Triumph of the lean production system", *Sloan Management Review* 30(1), 41-52.
- [Ladas2009]: Ladas, C. (2009). *Scrumban*, Modus Cooperandi Press (January 12, 2009).
- [Landis2011]: Sean Landis. 2011. Agile Hiring. Artima, Inc.
- [Laney2001]: Laney, Douglas. 2001. “3D Data Management: Controlling Data Volume, Velocity, and Variety.” <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf>. Accessed 2016-09-05.
- [Larman2002]: Larman, C. (2002), *Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process*, Prentice Hall PTR, Upper Saddle River, NJ
- [Larman2009]: Larman, C. & Bodde, V. (2009), *Scaling Lean & Agile Developments: Thinking and Organizational Tools for Large-Scale Scrum*, Addison-Wesley, Upper Saddle River, NJ
- [Leffingwell2010]: Leffingwell, D. (2010), *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, Pearson Education
- [Liker2004]: Liker, J. K. (2004), *The Toyota way : 14 management principles from the world's greatest manufacturer*, McGraw-Hill, New York
- [Limoncelli2014]: Limoncelli, T. A.; Chalup, S. R. & Hogan, C. J. (2014), *The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems*, Vol. 2, Pearson Education
- [Lins2016]: Lins, S., Grochol, P., Schneider, S., & Sunyaev, A. (2016). Dynamic Certification of Cloud Services: Trust, but Verify! IEEE Security & Privacy, 14(2), 66–71. <http://doi.org/10.1109/MSP.2016.26>
- [Lockwood2009]: Lockwood, Thomas. 2009. Design Thinking: Integrating Innovation, Customer Experience, and Brand Value. New York, N.Y.: Allworth Press - Allworth Communications.
- [Loeliger2009]: Loeliger, J. (2009). *Version control with Git*. Beijing ; Sebastopol, CA, O'Reilly.
- [Lucas2014]: Lucas, S. (2014). Nordstrom's awesome employee handbook is a myth. Retrieved June 29, 2016, from <http://www.cbsnews.com/news/nordstroms-awesome-employee-handbook-is-a-myth/>
- [Madachy2008]: Madachy, R. J. (2008). *Software process dynamics*. Hoboken, NJ Piscataway, NJ, Wiley; IEEE Press.
- [Malan2005]: Malan, Ruth, and Dana Bredemeyer. 2005. “Enterprise Architecture as Strategic Differentiator.” Cutter Consortium Enterprise Architecture Advisory Service Executive Report 8 (6).
- [Malan2010]: Malan, Ruth, and Dana Bredemeyer. 2010. “The Art of Change: Fractal and Emergent.” Cutter Consortium Enterprise Architecture Advisory Service Executive Report 13 (5).

- [Marks2014]: Marks, Howard. 2014. "Code Spaces: A Lesson In Cloud Backup." Network Computing. <http://www.networkcomputing.com/cloud-infrastructure/code-spaces-lesson-cloud-backup/314805651>. Accessed 2016-09-28.
- [McAdam2003]: McAdam, John. 2003. "Information Technology Measurements." In *Chargeback and IT Cost Accounting*, ed. Terence A Quinlan. Santa Barbara, CA: IT Financial Management Association, 90–91.
- [McCrory2010]: McCrory, Dan. 2010. "Data Gravity – in the Clouds." McCrory's Blog. <https://blog.mccrory.me/2010/12/07/data-gravity-in-the-clouds/>. Accessed 2016-09-01.
- [Meyer2013]: Meyer, N. Dean. 2013. Internal Market Economics: Practical Resource-Governance Processes Based on Principles We All Believe in. Dansbury, CT: NDMA Publishing.
- [Millotat1992]: Millotat, Christian. 1992. "Understanding the Prussian-German General Staff System." Carlisle Barracks, PA. <http://www.dtic.mil/cgi/tr/fulltext/u2/a249255.pdf>. Accessed 2016-10-04,
- [Minick2012]: Minick, E. (2012), "A DevOps Toolchain: There and back again", Slideshare.net, <http://www.slideshare.net/Urbancode/building-devops-toolchain>
- [Mintzberg1983]: Mintzberg, H. (1983). *Structure in fives : designing effective organizations*. Englewood Cliffs, N.J., Prentice-Hall.
- [Moeller2013]: Moeller, Robert R. 2013. Executive's Guide to IT Governance: Improving Systems Processes with Service Management, COBIT, and ITIL. Hoboken, New Jersey: John Wiley & Sons, Inc.
- [Moody2009]: Moody, Dan. 2009. "The 'Physics' of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering." Journal Article. IEEE Transactions on Software Engineering 35 (5): 756–78.
- [Moore2014]: Moore, Geoffrey. 2014. Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers. 3rd ed. New York, N.Y.: HarperCollins Publishers, Inc.
- [Munroe2013]: Munroe, Randall. 2013. "FedEx Bandwidth." What If? <http://what-if.xkcd.com/31/>. Accessed 2016-09-01
- [Murphy2007]: Murphy, Jacques. 2007. "Where Should Product Management Report?" Pragmaticmarketing.com. <http://pragmaticmarketing.com/resources/where-should-product-management-report>. Accessed 2016-09-14.
- [Narayam2015]: Narayam, S. (2015). Agile IT organization design: for digital transformation and continuous delivery, Pearson Education Inc.
- [NationalCourt2016]: The National Court Rules Committee. 2016. Federal Rules of Civil Procedure. <https://www.federalrulesofcivilprocedure.org/>.
- [NIST1993]: NIST. 1993. "Integration Definition for Function Modeling (IDEF0)." http://www.idef.com/idefo-function_modeling_method/.
- [Nordstrom2015]: Nordstrom, Inc. 2015. "Code of Business Conduct and Ethics." <http://investor.nordstrom.com/phoenix.zhtml?c=93295&p=irol-govconduct>. Accessed 2016-06-29.
- [Nygard2007]: Nygard, M.T., 2007. *Release it! : design and deploy production-ready software*. The pragmatic programmers, Raleigh, N.C.: Pragmatic Bookshelf. xvi, 350 p.
- [OASIS2013]: OASIS (2013), "Topology and Orchestration Specification for Cloud Applications

Version 1.0 (TOSCA)", <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>

- [Ohno1988]: Ohno, T. (1988). *Toyota production system : beyond large-scale production*, Productivity Press, Cambridge, Mass.
- [Olson2013]: Olson, Elizabeth. 2013. "Microsoft, GE, and the Futility of Ranking Employees." Fortune (November 18, 2013). <http://fortune.com/2013/11/18/microsoft-ge-and-the-futility-of-ranking-employees/>.
- [Opelt2013]: Opelt, A., B. Gloger, et al. (2013). *Agile contracts : creating and managing successful projects with Scrum*.
- [Open2009]: The Open Group. (2015). The Open Group Architectural Framework (TOGAF), Version 9 (Report). Open Group, The. Retrieved from <http://www.opengroup.org/togaf/>
- [Open2012]: Open Group, The. 2012. "Archimate 2.1 Specification." Standard. <http://pubs.opengroup.org/architecture/archimate2-doc/toc.html>.
- [Open2015]: Open Group, The. 2015. "IT4IT Standard." Open Group, The. <http://www.opengroup.org/it4it/>.
- [Osterwalder2010]: Osterwalder, Alexander, and Yves Pigneur. 2010. *Business Model Generation*. Wiley, 280. <http://www.businessmodelgeneration.com/canvas>.
- [Osterwalder2014]: Osterwalder, Alexander, Yves Pigneur, Greg Bernarda, and Alan Smith. 2014. *Value Proposition Design*. Hoboken, N.J.: John Wiley & Sons, Inc.
- [Padua2015]: Padua, Sydney. 2015. The Thrilling Adventures of Lovelace and Babbage: The (Mostly) True Story of the First Computer. New York: Random House.
- [Patton2014]: Patton, J., 2014. *User story mapping : discover the whole story, build the right product*. First edition. ed. xliv, 276 pages.
- [Peck2016]: Peck, Claude. 2016. "U Expert Tells How 'Design Thinking' Can Solve Society's Big Problems." Minnesota Star Tribune, July 16.
- [PMI2013]: Project Management Institute, 2013. A guide to the project management body of knowledge (PMBOK guide). Fifth edition.
- [Poppendieck2007]: Poppendieck, M. & Poppendieck, T. D. (2007), *Implementing lean software development : from concept to cash*, Addison-Wesley, London
- [Poppendieck2003]: Poppendieck, M. & Poppendieck, T. D. (2003), *Lean Software Development: An Agile Toolkit*, Addison Wesley, Boston
- [Porter1998]: Porter, Michael E. 1998. *Competitive Advantage: Creating and Sustaining Superior Performance : With a New Introduction*. 1st Free P. New York: Free Press. <http://www.loc.gov/catdir/bios/simon051/98009581.html>.
- [Portny2013]: Portny, S. (2013). Project Management for Dummies. Hoboken, New Jersey, John Wiley & Sons.
- [Puppet2015]: Puppet Labs. 2015. "2015 State of DevOps Report."
- [Quinlan2003]: Quinlan, Terence A. 2003. *Chargeback and IT Cost Accounting*. ed. Terence A Quinlan. Santa Barbara, CA: IT Financial Management Association.
- [Racynski2008]: Racynski, Bob, and Bill Curtis. 2008. "Software Data Violate SPC's Underlying Assumptions." IEEE Software 25 (3): 49–51.

- [Ran2001]: Ran, Alexander. 2001. "Fundamental Concepts for Practical Software Architecture." In Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM. doi:10.1145/503209.503269.
- [Reinertsen1997]: Reinertsen, D. G. (1997), *Managing the design factory: a product developer's toolkit*, Free Press, New York ; London
- [Reinertsen2009]: Reinertsen, D. G. (2009), *The principles of product development flow: second generation lean product development*, Celeritas, Redondo Beach, Calif.
- [Richardson2010]: Richardson, G.L., 2010. Project Management Theory and Practice, Boca Raton: Auerbach Publications, Taylor & Francis Group.
- [Ries2011]: Ries, E. (2011), *The lean startup : how today's entrepreneurs use continuous innovation to create radically successful businesses*, Crown Business, New York
- [Rigby2016]: Rigby, Darrell K., Jeff Sutherland, and Hirotaka Takeuchi. 2016. "Embracing Agile." Harvard Business Review, no. May. <https://hbr.org/2016/05/embracing-agile>.
- [Rogers2003]: Rogers, Everett. 2003. Diffusion of Innovations. 5th ed. New York, N.Y.: Free Press - Simon & Schuster, Inc.
- [Ross2006]: Ross, Jeanne W, Peter Weill, and David Robertson. 2006. Enterprise Architecture as Strategy : Creating a Foundation for Business Execution. Book. Boston, Mass.: Harvard Business School Press. Table of contents <http://www.loc.gov/catdir/toc/ecip0611/2006010226.html>.
- [Root1916]: Root, Elihu. 1916. The Military and Colonial Policy of the United States: Addresses and Reports. Cambridge, Mass.: Harvard University Press. http://www.shsu.edu/~his_ncp/RootGS.html. Accessed 2010-10-4.
- [Rother2003]: Rother, Mike, and John Shook. 2003. "Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA [Spiral-Bound]." Lean Enterprise Institute. doi:10.1109/6.490058.
- [Rother2010]: Rother, M. (2010). *Toyota kata: managing people for improvement, adaptiveness, and superior results*. New York, McGraw Hill.
- [Royce1970]: Royce, W. "Managing the Development of Large Software Systems." in Proc. IEEE WESCON. 1970. Los Angeles: IEEE.
- [Rozovsky2015]: Rozovsky, J. (2015). The five keys to a successful Google team. re:Work. Retrieved from <https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/>. Accessed 2016-09-21.
- [Rubin2012]: Rubin, K.S., 2012. *Essential Scrum : a practical guide to the most popular agile process*, Upper Saddle River, NJ: Addison-Wesley. xliii, 452 p.
- [Rummel1995]: Rummler, Geary A, and Alan P Brache. 1995. *Improving Performance: How to Manage the White Space on the Organization Chart*. The Jossey-Bass Management Series. 2nd ed. San Francisco, CA: Jossey-Bass. <http://www.loc.gov/catdir/toc/wiley041/94048105.html>.
- [Schlarman2008]: Schlarman, Steve. 2016. "Developing Effective Policy, Procedure, and Standards." www.disaster-Resource.com. Accessed June 30 2016. http://www.disaster-resource.com/articles/07p_106.shtml.
- [Schwaber2002]: Schwaber, Ken. 2002. Agile Software Development with Scrum. Upper Saddle River, N.J.: Prentice Hall.

- [Schwaber2007]: Schwaber, Ken. 2007. *The Enterprise and Scrum*. Redmond, Wash: Microsoft Press.
- [Schwaber2013]: Schwaber, K. (2013). unSAFE at any speed. URL: <https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/>. Ken Schwaber's Blog: Telling It Like It Is. 2016.
- [Scotland2010]: Scotland, K. (2010), "Defining the Last Responsible Moment", <http://availagility.co.uk/2010/04/06/defining-the-last-responsible-moment>
- [Sells1957]: Sells, S. B., Fixott., Richard S. (1957). "Evaluation of Research on Effects of Visual Training on Visual Functions." *Am J Ophthal* 44(2): 230-236.
- [Shannon1949]:Shannon, Claude Elwood, and Warren Weaver. 1949. The Mathematical Theory of Communication. Urbana,: University of Illinois Press.
- [Sharp2009]: Sharp, A. and P. McDermott (2009). *Workflow modeling: tools for process improvement and applications development*. Second edition. Boston, Artech House.
- [Shortland2012]: Shortland, A. & Lei, M. (2012), "Using Rundeck and Chef to build DevOps Toolchains", <http://dev2ops.org/2012/05/using-rundeck-and-chef-to-build-devops-toolchains-at-chefcon/>
- [Sigler2014]: Sigler, Eric. 2014. "So, What Is ChatOps? And How Do I Get Started?" Pagerduty.Com. <https://www.pagerduty.com/blog/what-is-chatops/>.
- [Silverston2001]:Silverston, Len. 2001. The Data Model Resource Book Vol 1: A Library of Universal Data Models for All Enterprises. Book. Rev. ed. New York ; Chichester: Wiley.
- [Silverston2001a]:Silverston, Len. 2001a. The Data Model Resource Book Vol 2: A Library of Universal Data Models by Industry Types. Book. Rev. ed. New York ; Chichester: Wiley.
- [Silverston2008]: Silverston, L. (2008), *The data model resource book Vol 3: Universal patterns for data modeling*, Wiley, Indianapolis, Ind.
- [Sirkia2013]: Sirkiä, Rami, and Maarit Laanti. 2013. Lean and Agile Financial Planning. <http://www.scaledagileframework.com/original-whitepaper-lean-agile-financial-planning-with-safe/>.
- [Simon1988]: Simon, Herbert A. 1988. "The Science of Design: Creating the Artificial." *Design Issues* 4 (1/2). <http://www.jstor.org/stable/1511391>.
- [Sims2012]: Sims, C.J., Hillary Louise, 2012. *Scrum: a Breathtakingly Brief and Agile Introduction*. Dymaxicon.
- [Smith1991]: Smith, P. G. and D. G. Reinertsen (1991). *Developing products in half the time : new rules, new tools* New York ; London, Van Nostrand Reinhold.
- [Smith1998]: Smith, P. G. and D. G. Reinertsen (1998). *Developing products in half the time : new rules, new tools (2nd edition)* New York ; London, Van Nostrand Reinhold.
- [Spinellis2015]: Spinellis, D. (2015). "Extending Our Field's Reach." *IEEE Software*: 4-6. December 2015.
- [Sterman2000]: Sterman, J., 2000. *Business dynamics : systems thinking and modeling for a complex world* Boston: Irwin/McGraw-Hill. xxvi, 982 p.
- [Sussna2015]: Sussna, Jeff. 2015. Designing Delivery: Rethinking IT in the Digital Service Economy. O'Reilly Publications.

- [Treacy1997]: Treacy, Michael, and Fred Wiersema. 1997. *The Discipline of Market Leaders: Choose Your Customers, Narrow Your Focus, Dominate Your Market*. New York, N.Y.: Basic Books - Perseus Books Group.
- [TSO2011a]: The Stationery Office. 2011. *Information Technology Infrastructure Library ITIL Service Strategy: 2011 Edition*. Norwich, U.K: The Stationery Office.
- [TSO2011b]: The Stationery Office. 2011. *Information Technology Infrastructure Library ITIL Service Design: 2011 Edition*. Norwich, U.K: The Stationery Office.
- [TSO2011c]: The Stationery Office. 2011. *Information Technology Infrastructure Library ITIL Service Transition: 2011 Edition*. Norwich, U.K: The Stationery Office.
- [Sutherland2014]: Sutherland, J.V., 2014. *Scrum: the art of doing twice the work in half the time*. Crown Business. First Edition. ed. viii, 248 pages.
- [Sutton2014]: Sutton, Robert I.;, and Huggy Rao. 2014. *Scaling up Excellence: Getting to More without Settling for Less*. Crown Business/Random House.
- [Thompson2014]: Thompson, L. (2014), "Hitchhikers Guide to OpenStack Toolchains", <https://www.openstack.org/assets/presentation-media/Hitchhikers-Guide-to-OpenStack-Toolchains.pdf>
- [Tidwell2006]: Tidwell, J. (2006). *Designing Interfaces*. Sebastopol, CA: O'Reilly Media, Inc.
- [Tikka2013]: Tikka, A., 2013. "Coordination Chaos." Slideshare deck. <http://www.slideshare.net/gosei/coordination-chaos?related=1> Accessed: 2015.10.25.
- [Traynor2016]: Traynor, Des. 2016. "Focus on the Job, Not the Customer." Inside Intercom. <https://blog.intercom.com/when-personas-fail-you/>. Accessed 2016-09-18
- [Tufte2001]: Tufte, E. R. (2001). *The Visual Display of Quantitative Information*. *The Visual Display of Quantitative Information* (Vol. 4).
- [Ulrich2010]: Ulrich, William, and Neal McWhorter. 2010. *Business Architecture: The Art and Practice of Business Transformation*. Book. Tampa, Florida: Meghan-Kiffer.
- [Uptime2014]: Uptime Institute. 2014. "Explaining the Uptime Institute's Tier Classification System." Uptime Institute Journal. <https://journal.uptimeinstitute.com/explaining-uptime-institutes-tier-classification-system/>. Accessed 2016-07-04.
- [Uptime2016]: Uptime Institute. 2016. "Tier Certification Tiers Is the Global Language of Data Center Performance Tier Certification Is Worldwide Credibility." https://uptimeinstitute.com/uptime_assets/2edec7f3207b2802cf5fad3ad50d85947e69a62ab7d4230c0f7be3ebdbfd3b2c-00006.pdf. Accessed 2016-07-04.
- [Venezia2014]: Venezia, Paul. 2014. "Murder in the Amazon Cloud." InfoWorld. <http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html>. Accessed 2016-07-04
- [Venkatraman2013]: Venkatraman, Archana. 2013. "2e2 Datacentre Administrators Hold Customers' Data to £1m Ransom." ComputerWeekly.com. <http://www.computerweekly.com/news/224017744/2e2-datacentre-administrators-hold-customers-data-to-1m-ransom>. Accessed 2016-07-04
- [Vergun2015]: Vergun, D. (2015). Toxic leaders decrease Soldiers' effectiveness, experts say. Retrieved July 13, 2016, from

https://www.army.mil/article/157327/Toxic_leaders_decrease_Soldierseffectivenessexperts_say

- [vonNeumann1947]: von Neumann, John, and Herman H. Goldstine. 1947. "Planning and Coding of Problems for an Electronic Computing Instrument." Princeton N.J. <https://library.ias.edu/files/pdfs/ecp/planningcodingof0103inst.pdf>. Accessed 2016-10-10.
- [Westerman2001]: Westerman, Paul. 2001. Data Warehousing: Using the Wal-Mart Model. San Diego: Academic Press.
- [WFMC2010]: WFMC. 2010. "Adaptive Case Management." <http://www.xpdl.org/nugen/p/adaptive-case-management/public.htm>. Accessed 2016-06-04.
- [Whittaker2012]: Whittaker, J.A., J. Arbon, and J. Carollo, 2012, *How Google tests software*, Upper Saddle River, NJ: Addison-Wesley. xxvii, 281 p.
- [Womack2003]: Womack, J. P. & Jones, D. T. (2003), *Lean thinking: banish waste and create wealth in your corporation*, Free Press, New York
- [Womack1990]: Womack, J. P.; Jones, D. T. & Roos, D. (1990), *The machine that changed the world : based on the Massachusetts Institute of Technology 5-million dollar 5-year study on the future of the automobile*, Rawson Associates, New York
- [Zachman1987]: Zachman, John. 1987. "Zachman Framework." IBM Systems Journal 26 (3): 276–92.