Agile IT Management: From Startup to Enterprise (Collaborative Draft)

Table of Contents

Note: The README is temporarily included in the actual book build. This will likely be removed for final publication.

README — Agile IT management: from startup to enterprise

Note, if you came to this via the Github repository (https://github.com/dm-academy/aitm), you might want to look at the content which is now being rendered to Github Pages (http://dm-academy.github.io/aitm/). Same stuff, but more navigable and the whole book will be there. If you're already on Github Pages (http://dm-academy.github.io/aitm/) never mind.

7/23/2015 Work in progress: I am moving most of this README to the preface (http://dm-academy.github.io/aitm/#_preface), but there will remain a placeholder w/pointers here.

Status as of 7/16/2015: I have ~60,000 words in Scrivener that I am starting to transfer to Github. This is not a simple conversion, the material needs further rewriting as I do this. But this project is by no means starting from scratch. Stop back in a few weeks and you'll see actual content.

Updated status as of 8/6/2015: Have a complete file system and placeholder files, first 2 chapters are reasonable first drafts, working on Chapter 3.

Views on the book

There are three main views on the book:

- The Student Edition (http://dm-academy.github.io/aitm/)
- The Instructor's Edition (http://dm-academy.github.io/aitm/aitm-instructor.html), with commentary, background, and additional resources intended for instructors using the book in their class
- The Collaborative Draft (http://dm-academy.github.io/aitm/aitm-collaborator.html), including the content of both editions, work in process, and collaborative discussion

All are generated from the same source files. Include macros are used to include the additional instructor commentary.

Participating

I need reviewers and content contributors. I especially appreciate well-informed sidebars on specific topics and will give full in-text authorial credit for such. Or, you can help with the body of the text. I also want to cultivate an ecosystem of labs, but that is a different story.

My desire is that interested parties contribute to this project via standard Github techniques. I realize this places a barrier for some otherwise qualified contributors. However, I believe that **source control is a singularly important practice**; it is the constraint which enables much of the benefits of Agile.

Learning Github is a good use of your time, if you seek to teach the next generation of IT practitioners. They are heading out into a world of "infrastruture as code" and pervasive use of source control. Github portfolios are increasingly selection criteria in the hiring process for IT professionals.

(Of course, if you have a nice standalone sidebar, or a **few** random edits, send them on to me however and I can work them in.)

In terms of an Agile medium, the format of the book on Github will allow for better "random access" reviewing. This I think will be an improvement over the standard "read a big batch of my work please." I'll point to specific sections as they are completed, and as discussions progress on Twitter & other channels, an approach I think is more agile.

I will say more about the labs elsewhere, but I have found that developing good labs is extraordinarily labor-intensive. I intend that each chapter have a solid lab (or multiple alternatives) and would greatly value any contributions that align with the book's progression.

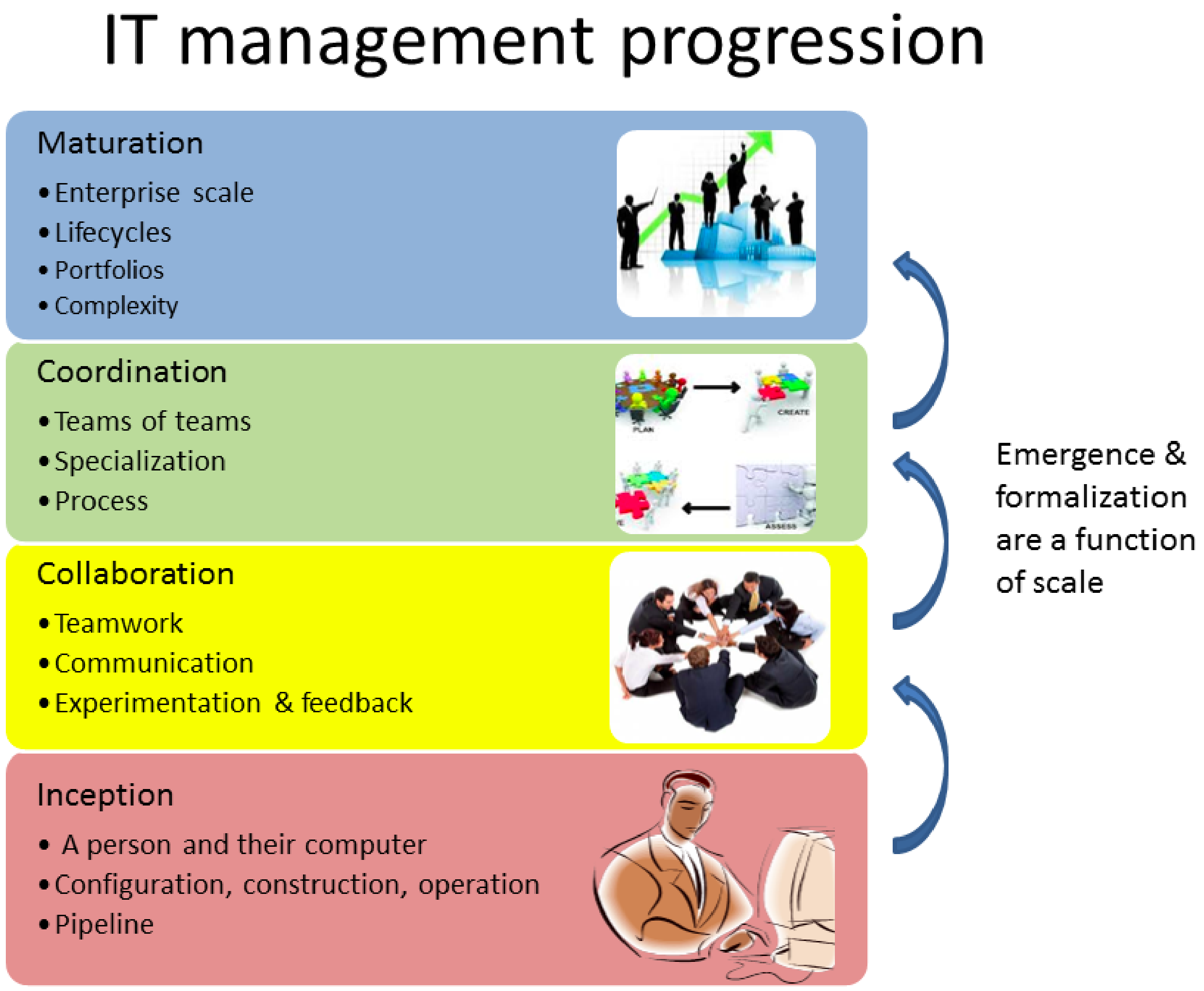
The road ahead

I have not decided on a publishing channel. I am experimenting with LeanPub, but the final decision is pending. The work would always be free here, but I may put additional formatting work in and sell the resulting value-added product. I am considering setting up a nonprofit of some sort to further the work. And I am still considering the licensing options. Interested in any ideas along these lines.

My stance on these issues depends directly on other people making substantial contributions. If that does not happen, I will consider other, more proprietary models, although I remain concerned for educational access and am not inclined to publish this solely as a $150.00 glossy textbook.

If you have comments or questions, please log a Github issue (https://github.com/dm-academy/aitm/issues) on this repository. Or drop me a line at char AT dm-academy DOT com.

End of README, beginning of actual book build.



Preface

Welcome to Agile IT Management, the first general, survey-level text on IT management written from an Agile and Lean IT perspective.

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 Agile

Both the IT industry, and my understanding of it, have transformed dramatically in the past ten years, and the transformation has accelerated even in the last five. However, IT remains under-managed and poorly understood relative to peer functions in the enterprise. 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

The challenges of this legacy are an ongoing theme through this introductory text. 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.)

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

A primary goal of this book, as a textbook, is to prepare the student for an information technology career in industry, “industry” being broadly defined as both those industries that offer IT products, as well as industries that rely on IT 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."

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. We teach a broad variety 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 titled "IT Infrastructure" for historical reasons (derived from the "IT Infrastructure Library," or ITIL), ands was intended from its creation to cover the management of IT broadly. It is not limited to a narrower definition of "IT infrastructure" that would focus only on servers, operating systems, storage, networking, data centers, and the like. Rather it serves as a contextual course for the students in their in-depth studies of programming, databases, security, networking, and so forth.

Instructor's note

As of this writing (July 2015) I am looking forward to teaching my class a fourth 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, many of whom I count as friends. None of them feel well served by the current IT educational system.[1] "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 (http://www.meetup.com/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)* (http://http://www.amazon.com/Architecture-Patterns-Management-Resource-Governance/dp/0123850177)
. 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. (Yes, the title was far too long, due to an unsatisfactory compromise with the publisher.) 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 the above-mentioned "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 legitmate 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 communicating 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 Information Technology Infrastructure
(http://www.stthomas.edu/gradsoftware/programs/catalog/#d.en.116975)

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.

While my spring 2015 class was about half full, based on the older ITIL description, my fall 2015 section filled to capacity immediately when the new description was published.

Considering a 3rd edition

For the past three semesters I have 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 have become clearer and clearer throughout the 3 semesters I have taught to date. In particular, it had a strongly architectural approach, approaching the IT management problem as a series of views on a model (https://en.wikipedia.org/wiki/4%2B1_architectural_view_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 (https://github.com/StThomas-SEIS660), and had in fact had developed a reasonably sophisticated "DevOps in a Box" toolchain (the Calavera project

(https://github.com/CharlesTBetz/Calavera), which has attracted collaborators from the US, Spain, and Israel).

Hence this project.

End instructor's note

The 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

positioning this book vis a vis systems engineering, computer science, etc - is there some kind of matrix?

Omissions in this book:

- Big data
- Security

Current textbooks

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* (http://www.amazon.com/Introduction-Agile-Methods-Sondra-Ashmore/dp/032192956X)

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, 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.

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

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

Introduction

A process of emergence

In keeping with the entrepreneurial spirit that gave rise to the Agile movement and works like Ries' *The Lean Startup*, (http://www.amazon.com/dp/0307887898/) I am structuring this book around a progressive, evolutionary approach. Your journey through it reflects a process of emergence. Such processes are most often associated with founding and scaling a startup. There are many helpful books on this topic, such as:

- *Nail it then scale it* (http://www.amazon.com/Nail-then-Scale-Entrepreneurs-Breakthrough/dp/0983723605) by Furr and Ahlstrom
- *Scaling Up* (http://www.amazon.com/Scaling-Up-Companies-Rockefeller-Habits-ebook/dp/B00O5RR7QO/) by Harnish
- *Startup CEO* (http://www.amazon.com/Startup-CEO-Scaling-Business-Website/dp/1118548361) by Blumberg
- *The Lean Startup* (http://www.amazon.com/The-Lean-Startup-Entrepreneurs-Continuous/dp/0307887898/ref=pd_bxgy_14_img_y) by Ries

However, this is not a textbook (or course) on a full startup process. It remains IT-centric. **And, the book is also intended to be relevant to students entering directly into large, established enterprises.** In fact, because it progresses through the four contexts:

- Individual
- Team
- Small company
- Enterprise

It prepares you for working in companies at all stages of growth.

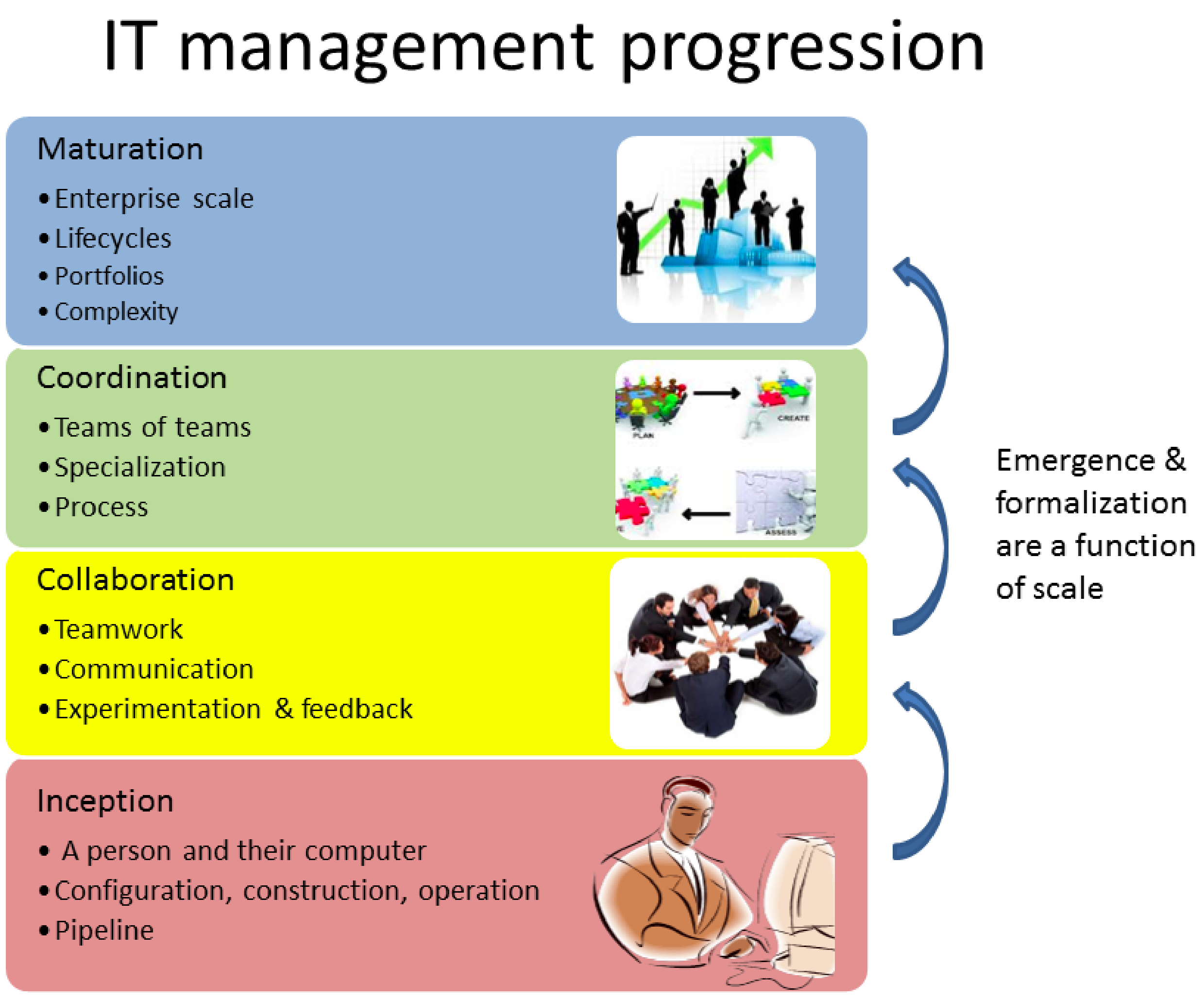
Whether you are in a startup, or on a journey within a larger, established organization, you will (hopefully) become aware as you progress of a broadening context:

- Other team members
- Customers
- Suppliers
- Sponsors
- Necessary non-IT capabilities (finance, legal, HR, sales, marketing, etc)
- Channel partners
- 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 you; my intent is to give you some useful tools.

This book's structure

Here is a conceptual illustration of an IT management progression:


IT management progression

Maturation
- Enterprise scale
- Lifecycles
- Portfolios
- Complexity

Coordination
- Teams of teams
- Specialization
- Process

Collaboration
- Teamwork
- Communication
- Experimentation & feedback

Inception
- A person and their computer
- Configuration, construction, operation
- Pipeline

Emergence & formalization are a function of scale

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:

A. **Inception**: The students are teamed as pairs of practitioners , each in a hypothetical startup, in a garage.

B. **Collaboration**: The students are in teams of 6-9, enough for specialization to emerge, but still intensely collaborative. A startup that has started to become viable.

C. **Coordination**: 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.

D. **Maturation**: Students are in groups of 4-5, representing the executive leadership of a large enterprise, concerned with IT portfolios, analytics, and the complex dynamics of running hetergeneous IT at massive scale.

Elaborating the above outline into chapters, we have:

I. **Inception**

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. **Collaboration**

4. *Product management* What exactly is it we are building? How do we better define it for a bigger team?
5. *Work management* How do we keep track and communicate at the most basic level?
6. *Operations management* How do we sustain this surprisingly fragile computing-based service, in its ongoing delivery of IT value?

III. **Coordination**

7. *Culture and organization* We’re getting big. How do we deal with this? How are we structured? Why this way and not that?
8. *Project and resource management* Work is becoming larger and more complex. We need the ability to track and execute larger segments of it.
9. *Process management* We have a structure. Work needs to flow across it.

IV. **Maturation**

10. *Architecture and governance* We need to understand the big picture of interacting lifecycles, reduce technical debt and redundancy, obtain better economies of scale, and cope with external forces (regulators, vendor partners, security adversaries, auditors) increasingly defining our options.
11. *Portfolio and analytics* We need to define our investment strategy based on a sound understanding of both business needs and technology limitations. We need to measure this massive IT estate and understand it as an overall dynamic system.

There are a wide variety of practices, techniques, and concerns that emerge at this point.

12. *Chaos, complexity, and the road ahead* No matter how we try, stuff happens, and it’s getting harder to cope with as the systems get bigger and more complex. Where will this all lead?

V. **Appendices**

1. *A review of IT frameworks and standards*
2. *Architectural depictions*
3. *Towards a theory of IT management*

The central advantage of this approach is that it is ideal for the new student. There really are no prerequisites for a course based on this text, although it can be a lot of work for those completely new to IT. 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.

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 twelth 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) 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 adopt ITIL or CMM before formalizing project management, that would indicate that chapters 8 and 9 should be flipped.

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…[2]

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.

End instructor’s note

Labs

note, this may be broken out into separate volume

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		Basic Linux command line	Individual
SECTION I: INCEPTION				
Chapter 1	IT value	Defining an IT product	Cloud and	2

			infrastructure as code	
Chapter 2	IT Infrastructure		Cloud and infrastructure as code	2
Chapter 3	Applications		Continuous delivery pipeline	2
SECTION II: COLLABORATION				
Chapter 4	Product management		Continuous delivery pipeline	6-8
Chapter 5	Work management		Ticketing and kanban	6-8
Chapter 6	Operations management	Service definition	Monitoring (Calavera + Nagios)	6-8
SECTION III: COORDINATION				
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		Open-source	>11 (full class)

	management		ITSM suite	
SECTION IV: MATURATION				
Chapter 10	Architecture & governance	Enterprise architecture & risk mgmt		5
Chapter 11	Portfolio and analytics	Portfolio simulation	Porfolio simulation with R?	5
Chapter 12	Complexity and chaos	Complex Incident Simulation		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.

- todo: should every chapter have an enterprise as well as startup reading?

Section I: Inception

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 partner. 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, the specific technologies employed in this books’ accompanying labs are based on Ubuntu Linux and git, the distributed version control system created by Linus Torvalds to facilitate Linux development.

```
[break labs & their discussion out into different book]
```

Chapter 1. IT Value

Introduction to Chapter 1

As noted at the outset, you are one or two people in 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 (whether you intend or not) 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?

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.

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 on line, 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 technology, some of it old, some of it new.

The screen on her cell phone represents information accessed and presented via a Model-View-Controller framework (https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller), implemented in the latest version of Javascript (https://developer.mozilla.org/en-US/docs/Web/JavaScript), running on an interpreter (https://en.wikipedia.org/wiki/Interpreter_(computing)) that would have taxed a mainframe (https://en.wikipedia.org/wiki/Mainframe_computer) thirty years ago. The communication with her bank's central systems is supported by 4G LTE (https://en.wikipedia.org/wiki/LTE_(telecommunication)) data which in turn relies on the high-volume IP backbone (https://en.wikipedia.org/wiki/Internet_Protocol) networks operated by the telecommunications carriers (http://searchnetworking.techtarget.com/definition/telecom-carrier), based on research into packet switching (https://en.wikipedia.org/wiki/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 (https://en.wikipedia.org/wiki/Middleware), crossing multiple network (https://en.wikipedia.org/wiki/Computer_network) control points. This middleware talks in turn to the customer demand deposit system which still runs on the mainframe.

The mainframe is now running the latest version of IBM's zOS (https://en.wikipedia.org/wiki/Z/OS) operating system (a direct descendant of OS/360 (https://en.wikipedia.org/wiki/OS/360_and_successors#MVT), one of the most significant operating systems in the history of computing (https://en.wikipedia.org/wiki/History_of_computing)). The "customer demand deposit" banking application running on the mainframe is still based on code written in the lowest level assembler (https://en.wikipedia.org/wiki/Assembly_language).

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 and, 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 (https://en.wikipedia.org/wiki/Cloud_computing) 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 (https://www.linux.com/).

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.

Discuss: How does information technology contribute to your enjoyment of life and experiences of value?

picture credits:

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

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

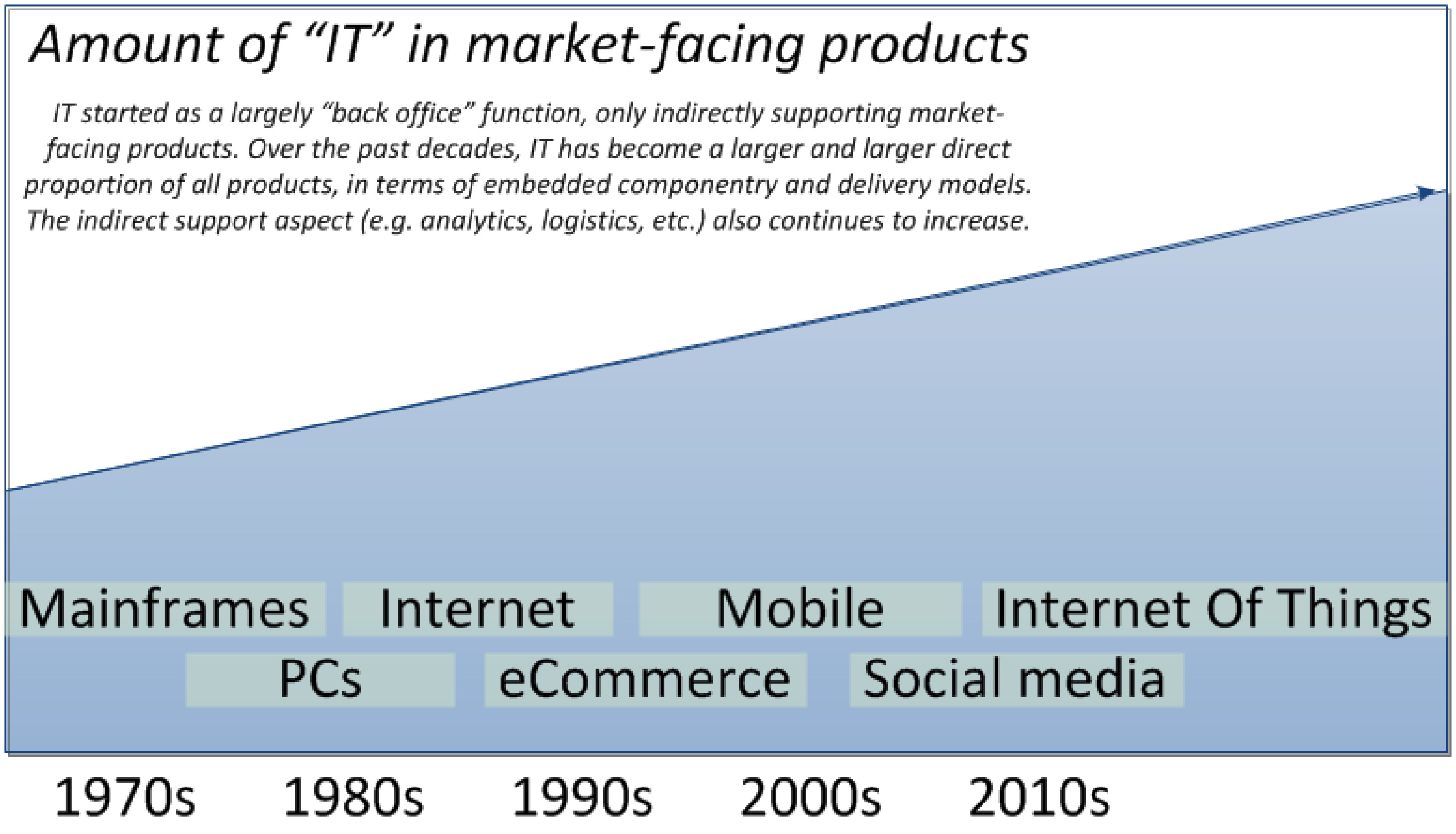
Defining Information Technology

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 (https://en.wikipedia.org/wiki/Claude_Shannon), Alan Turing (https://en.wikipedia.org/wiki/Alan_Turing), Alonzo Church (https://en.wikipedia.org/wiki/Alonzo_Church), John von Neumann (https://en.wikipedia.org/wiki/John_von_Neumann), and the other pioneers who defined the central problems of information theory (https://en.wikipedia.org/wiki/Information_theory), digital logic (https://en.wikipedia.org/wiki/Digital_electronics), computability (https://en.wikipedia.org/wiki/Computability), and computer architecture (https://en.wikipedia.org/wiki/Computer_architecture).



Information technology (”IT” for the rest of this book) increasingly permeates business operations and social interactions (figure XX). As it increases its pervasiveness, it becomes difficult to disentangle and analyze it. However, doing so remains essential. Without a line of sight back to its intellectual foundations, any technology risks becoming magic, and its users risk becoming a cargo cult (https://en.wikipedia.org/wiki/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 rec[1]reate 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.

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. There is also no doubt that “cargo cult thinking” is a useful cautionary metaphor.

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 vendors like HP vs IBM was no longer strategic. Looking to history, Carr argued that just as business no longer have “Vice Presidents for Electricity,” so businesses no longer need Chief Information Officers or dedicated IT departments.

Carr’s hypothesis has insight — there is no question IT is becoming pervasive — but 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 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 that are not comparable to simple electrical power.

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 this book is particularly concerned with.

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 routers), you are doing IT.

And regardless of where the 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.

Sidebar: Corporate function vs. problem domain

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 termf 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 through 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 is always a service, always a moment of truth - find from previous work

IT services, systems, and applications

Let’s examine our user’s value experience in more detail, without getting unnecessrily technical, and clarify some definitions along the way.

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

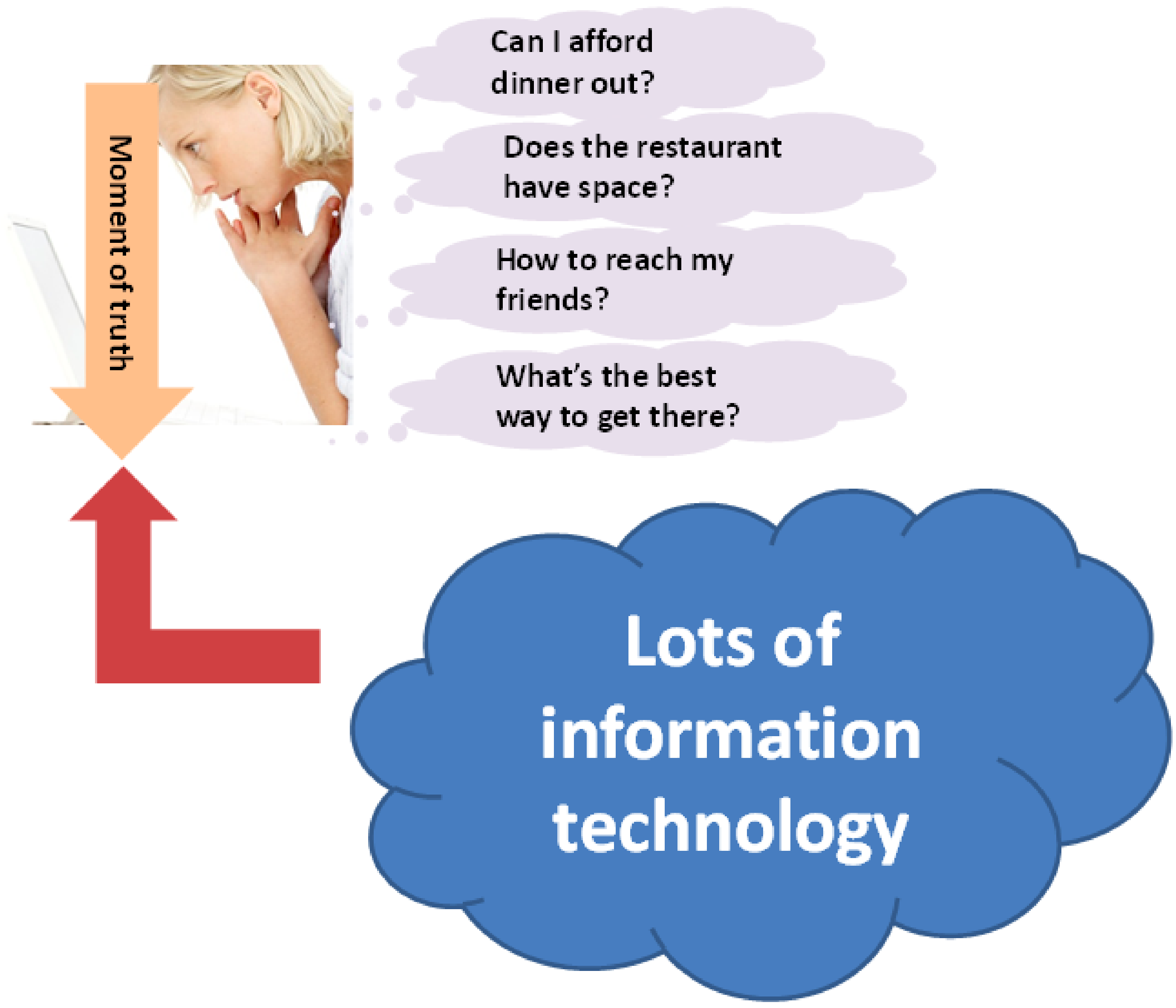
NOTE

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.


Can I afford dinner out?
Does the restaurant have space?
How to reach my friends?
What’s the best way to get there?
Moment of truth
Lots of information technology

In order to view her balance, our user is probably using an application downloaded from a "store" of applications made avaialble 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 screeen
- finally, displaying the information on the screen

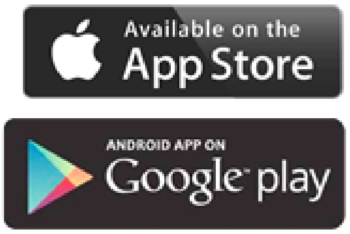
The application, or "app," downloaded to the phone plays a primary role, but is enabled by:

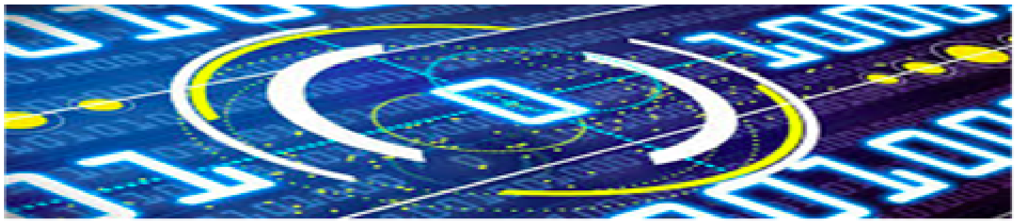
- the phone's operating system and associated services
- the phone's hardware
- the telecommunicatons infrastructure (cell phone towers, long distance fiber optic cables, switching offices, and much more)

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:

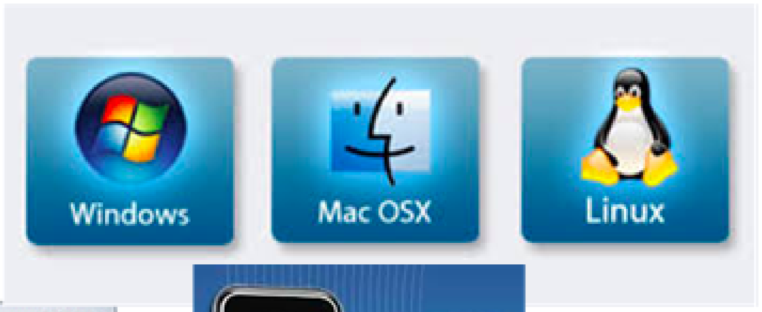
- Internet and middleware services to receive the request from the international network
- Application services to validate the user's identity and route the request to the appropriate handling service
- Data services to store the user's banking information (account identity and transactions) along with 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.

Mobile Banking
Savings
$3,287.36
$942.87

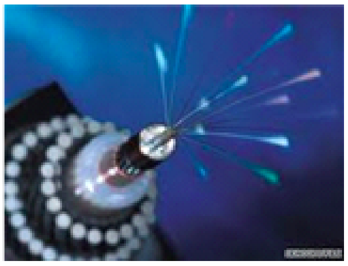

Available on the
App Store
ANDROID APP ON
Google play

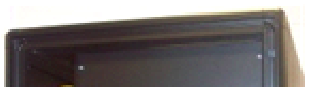


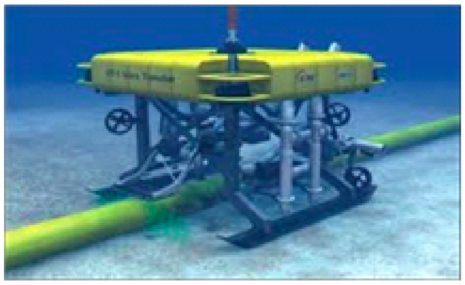
iOS
android


Windows
Mac OSX
Linux

WiFi

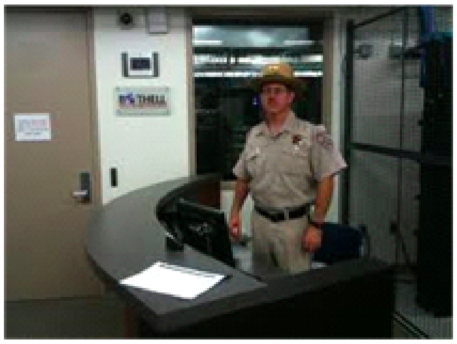


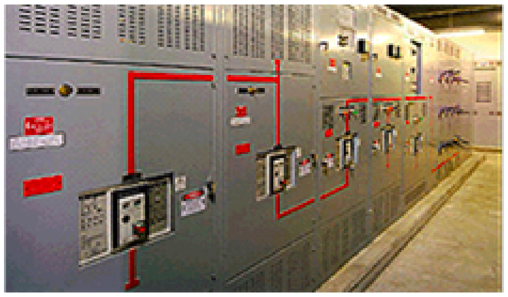




ORACLE

IBM





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 mantain 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.

This might seem obvious, but information technology has a well deserved reputation for being too complicated for end users. One can be trying to achieve something that should be simple, and find oneself too easily in a conversation about technical network settings.

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 long Figure 1.01.03 is and how it appears "stacked." Layered approaches to understanding IT are common; see Further Reading for useful references.

The IT service lifecycle

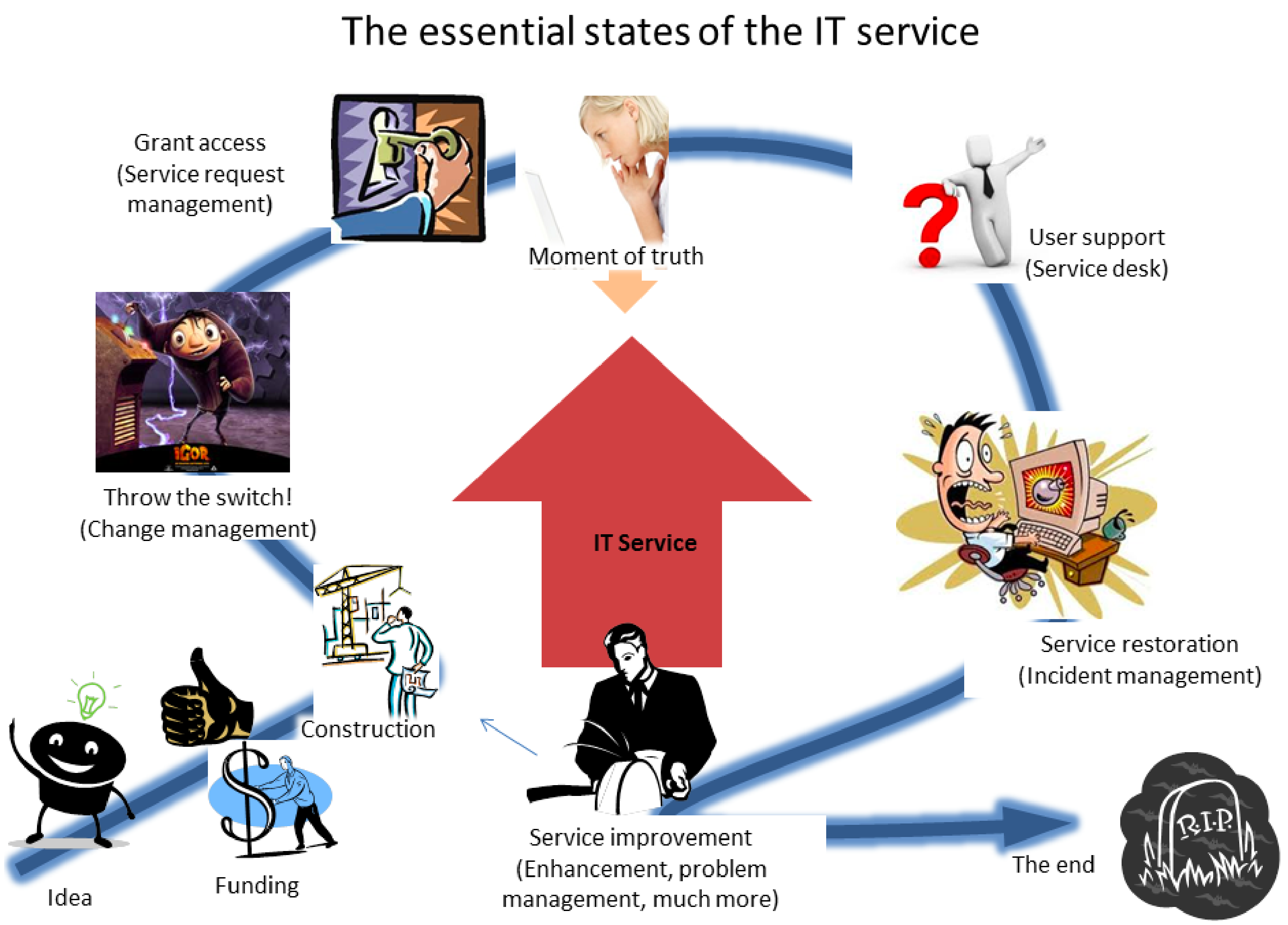
We've established that the IT service is based on a complex stack of technology, from local devices to global networks to massive data centers. Software and hardware are layered together in endlessly inventive ways to solve problems people did not even know they had ten years ago.

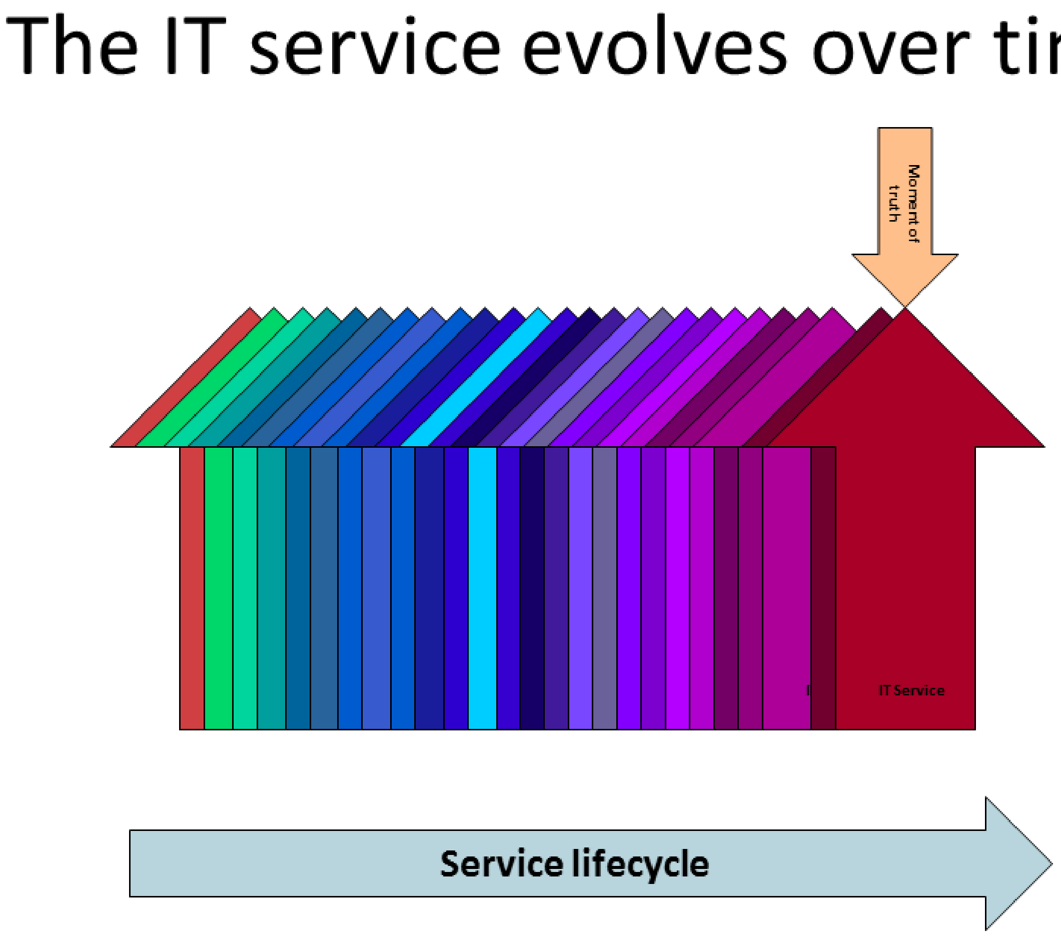
But these IT service systems must come from somewhere. They must be designed, built, and operated, and (as any smartphone user can testify) they are continually improved over time.

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

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, **they 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 systm is ON, there are still distinct changes in state when new features are deployed, or incorrect behavior ("bugs" or "defects") is 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 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 informaal 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,)


The essential states of the IT service
Grant access (Service request management)
Moment of truth
User support (Service desk)
Throw the switch! (Change management)
IT Service
Service restoration (Incident management)
Construction
Idea
Funding
Service improvement (Enhancement, problem management, much more)
The end
R.I.P.



The IT user and the IT customer

So, who paid for our user’s enjoyment? The bank and restaurant both had clear motivation for supporting a better on line 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 users 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.

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. The existence of global Internet connectivity relies on laws supporting utility easements and rights of way. Without a social infrastructure, social and collaborative technology has no meaning.

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

- discussions of IT stack
- *Code* (http://www.amazon.com/Code-Language-Computer-Hardware-Software/dp/0735611319), Charles Petzold
- *The Information,* (http://www.amazon.com/Information-History-Theory-Flood/dp/1400096235/ref=sr_1_1?s=books&ie=UTF8&qid=1438398170&sr=1-1&keywords=the+information) James Glieck
- *The Lean Startup* (http://www.amazon.com/Lean-Startup-Entrepreneurs-Continuous-Innovation-ebook/dp/B004J4XGN6/ref=sr_1_1?s=books&ie=UTF8&qid=1438398231&sr=1-1&keywords=the+lean+startup) , Ries

Chapter 2: Infrastructure Management

Introduction to Chapter 2

As mentioned in the section introduction, you cannot start writing code until you choose a programming language, which also has implications for how you ultimately deliver your IT-based product. (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 a decision 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.

Chapter summary

- Defining "IT infrastructure"
  - How an IT system **runs**, as opposed to what it **does**.
  - The ingredients (OS, language, services)
- IT infrastructure from a management point of view
- Physical vs virtual
- Cloud systems
- Servers vs containers vs services

Sidebar: Introducing Allspaw.

- Operations (not too much as that is the topic of Ch 6)
- Automation and infrastructure as code
  - Policy-based approaches
- The critical importance of source 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 source code control and why it applies to infrastructure management

Infrastructure overview

In the previous chapter, you developed a product idea and a proposed Minimum Marketable Feature. Some part of that Minimum Marketable Feature requires writing software, or at least configuring some IT-centric system. (IT being defined as in Chapter 1.)

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. 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 & 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. 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.

Sidebar: 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].

The interesting thing is that 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.

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.

From "physical" compute to Cloud

What do we even mean by "Cloud"?

First we need to understand what we mean by IT

- Computing cycles
- Memory & storage
- Communications

Those are the fundamentals

The idea of running IT as a utility service goes back years

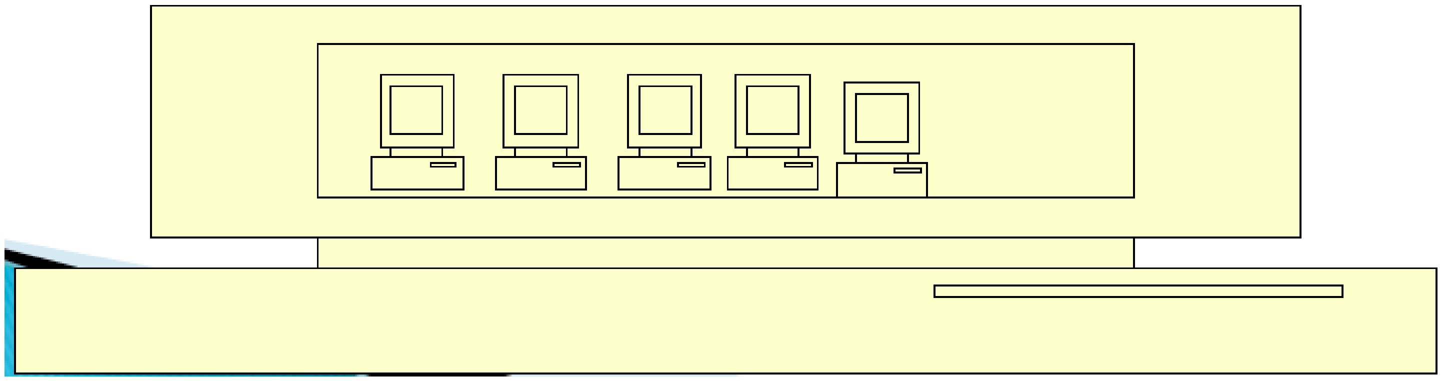
- Complexity has been the problem

Cloud origins

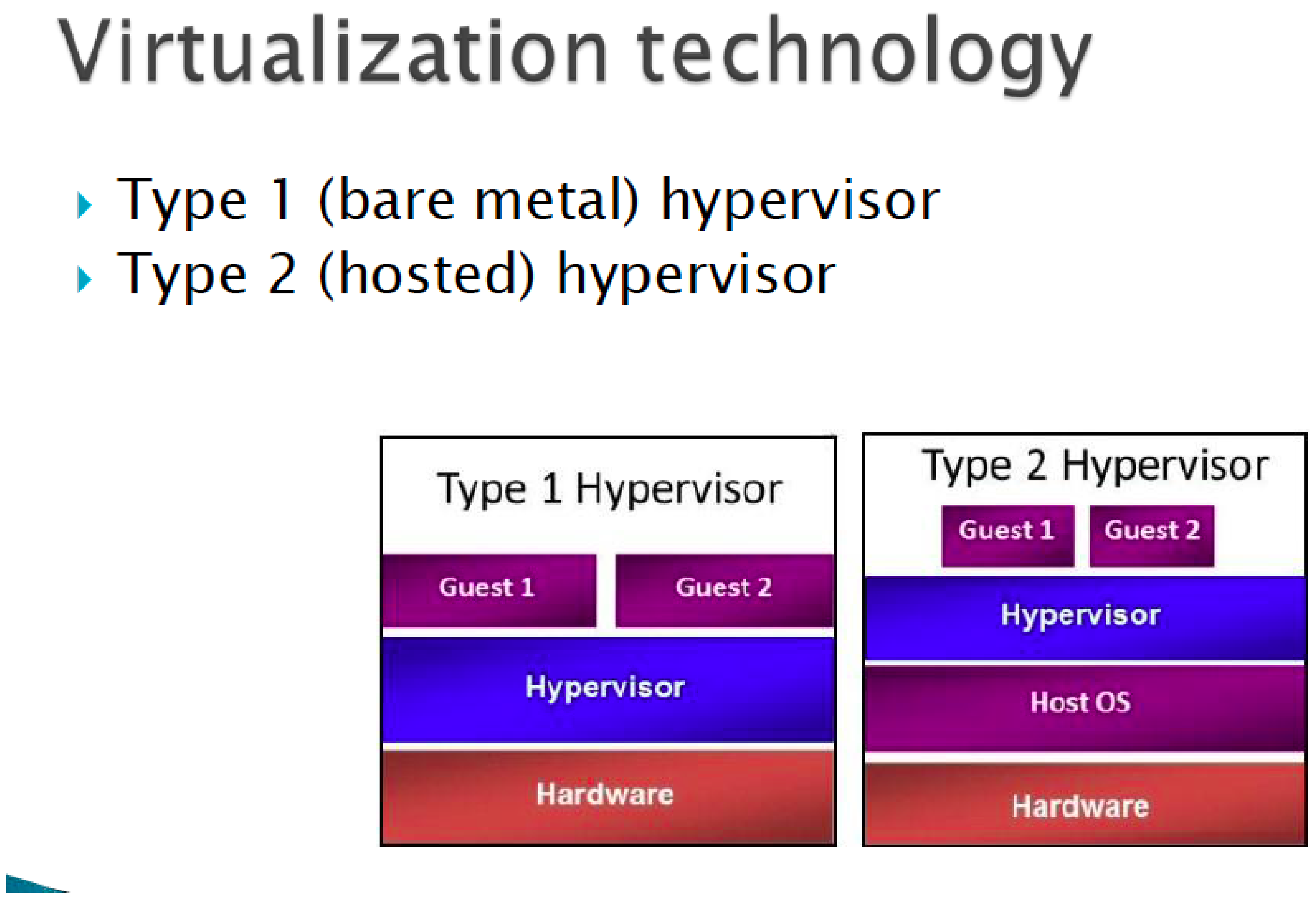
- Virtualization (we will discuss further)
- Managed hosting
- Web-scale IT
- “IT doesn’t matter”

Virtualizaton

- Essentially, a computer within a computer.



- There is a host system that can instantiate one or more guests.
    - The host system knows about the guests, but the guests do not know about the host (except as a peer system on the network, perhaps).
    - The physical resources allocated can vary according to demand
- The technical details are deep. Talk to your OS instructor.



Virtualization technology

- Type 1 (bare metal) hypervisor
- Type 2 (hosted) hypervisor

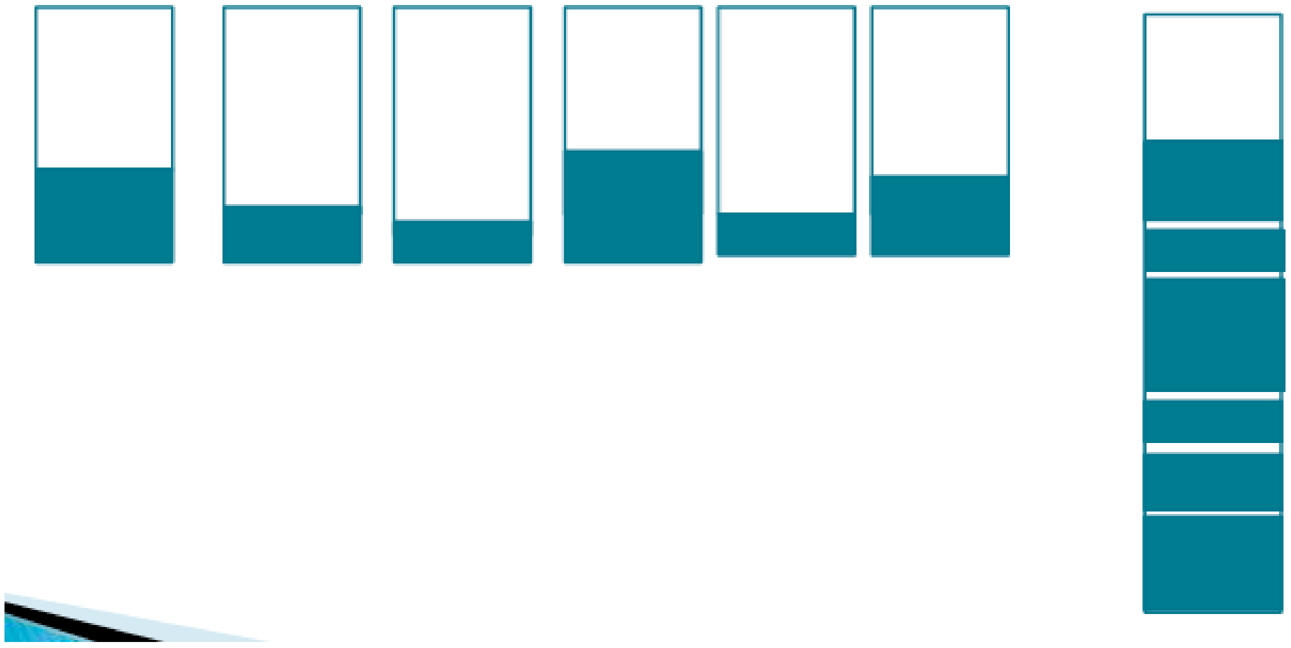
Did you know?

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 (IBM LPAR?).

- Multi-tenancy
  - Multi-tenancy is where multiple customers share physical resources that provide the illusion of being dedicated
  - The phone system has been multi-tenant ever since they got rid of party lines.

Virtualization vs containers

Virtualization benefits



Why virtualize?

- Great for heterogeneous workloads – lots of miscellaneous applications & services running on underutilized servers
- What about where the application is large and virtualization is mostly overhead?
  - Still may make sense
- Management consistency
- Ease of restoration
- Database professionals still don’t like it
  - “The database IS virtualization” they will say

Virtualizaiton vs cloud

- “Cloud is not just virtualization” many critics will say
- Financial model
- Tenancy model
- Provisioning model
- Services beyond raw compute, storage and network

- Self-service, API-driven

Traditional managed hosting vs cloud * Managed services: “your mess for less” * Cloud: “clean it up first”

Choosing infrastructure

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 X?”

As a startup, you have two major decisions to make in product development:

- What toolset should I use to create my product?
- What is my approach for exposing that product to the world? 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 is not the purpose of this book. However, this book has certain biases and 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.
    - continue to revisit these assumptions

This is where this textbook differs from many previous textbooks.

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. 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.

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 PHP, or Node, 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?
- Will this scale to a bigger codebase? Higher performance? Faster throughput of new features?

If you start to become uncomfortable with the answers, you should strongly consider investigating alternatives.

In Scrum-based development, they use the concept of a “spike” to represent effort whose outcome is not shippable product, but rather research and information. Consider thinking of things in this way.

https://www.scrumalliance.org/community/articles/2013/march/spikes-and-the-effort-to-grief-ratio

Iterative was not possible with earlier technical platforms.

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.’”

(Kim, Gene; Behr, Kevin ; Spafford, George (2013-01-10). The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win (p. 116). IT Revolution Press. Kindle Edition.)

Obviously (and as discussed in the Phoenix Project), “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

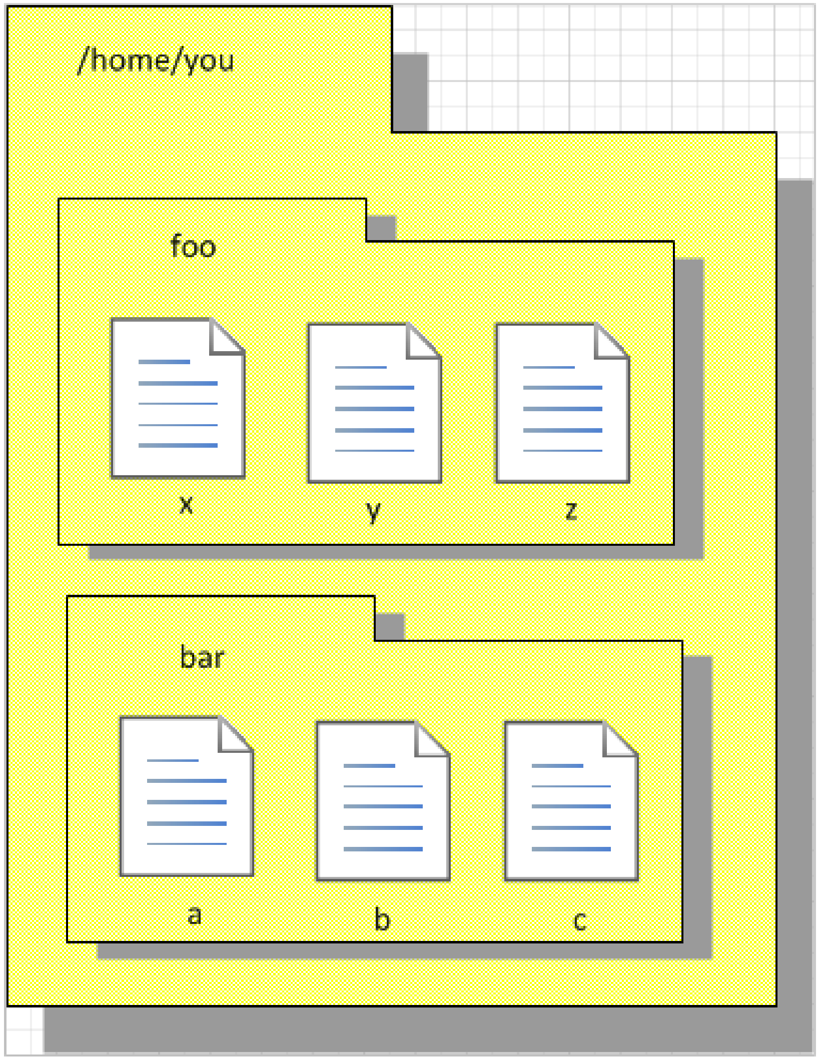
Note, the below section is illustrative only, and is not intended as a lab. The associated lab for this book goes into much more depth on these topics.

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 …

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

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.

From scripts to policies

Source control: the Agile foundation

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 source control (aka software configuration management) 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 “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 surrounding 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.

Thought: The concept of a commit is ur-metadata and the foundation of all IT management and governance.

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.)

- agile alliance map

So, what is source control?

Source control is like your file system, if it remembered all the changes you make to its contents, and could 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.

You can find many references to source control on the Internet and in books such as *Pro Git* by Scott Chacon and Ben Straub. 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:

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.

(Loeliger, Jon; McCullough, Matthew (2012-08-14). Version Control with Git: Powerful tools and techniques for collaborative software development (Kindle Locations 1520-1528). O’Reilly Media. Kindle Edition.)

```
Discussion of branching & merging?
```

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 given 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, I favor 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, 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.**

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

- Linux in a Nutshell
- Pro Git
- Limoncelli
- Allspaw

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" (draw on EE, a toaster is an "application" of AC current)
- How applications have been delivered over time
  - How installed
  - How experienced
- Basic application vs service vs software.
- Systems engineering (e.g. complex aerospace hardware/software systems) uses different terminology.
- Application lifecycle
- Earlier software development
  - "Waterfall" that wasn’t
  - Project mgmt history under McNamara
- Introduction to Agile proper
  - history, manifesto, etc

Sidebar: Introducing Kim & Humble

- Introduction to DevOps
  - 10 deploys a day, Phoenix Project, etc
- Application toolchain (source control introduced in prev chapter)
  - Requirements approaches (more on this in chapter 4)
  - Test-driven development
  - Continuous integration
  - Inspections and static analysis
  - Build tool
  - Package repo
- Trends in AD
  - Fast pace of innovation
  - Microservices
  - Re-use & external code

Learning objectives

- Define "application" as opposed to "infrastructure"
- Describe the full application lifecycle
- Define "Agile" in terms of software development
- Identify the major components of an end-to-end DevOps delivery pipeline.

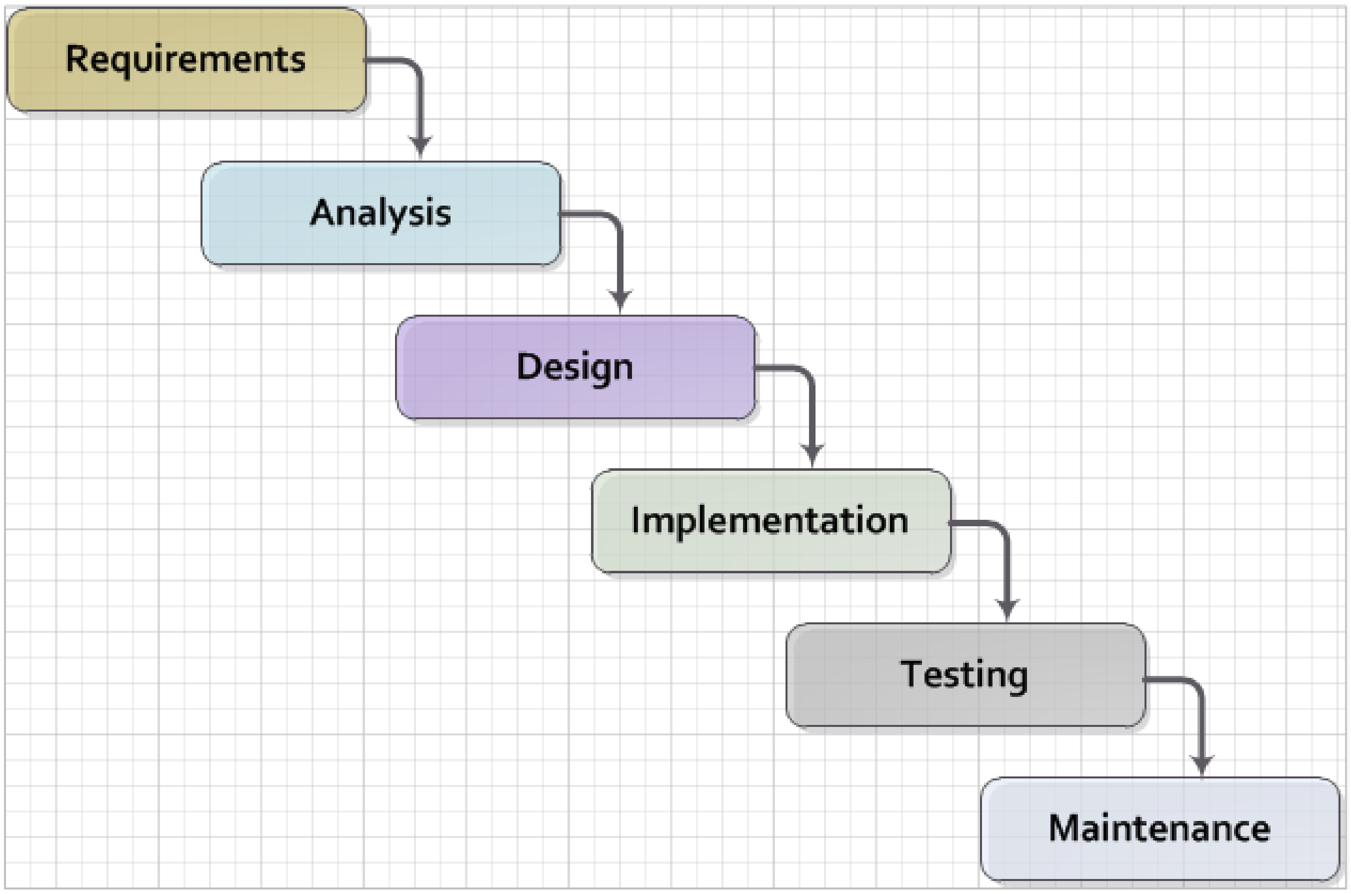
History of the Agile movement

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.)

We now understand that the product development process starts with a concept of requirement (whether we call it story, use case, or scenario is not important). But 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. (***cite Royce) Waterfall development as a term has become associated with a number of practices. The original illustration was similar to this:



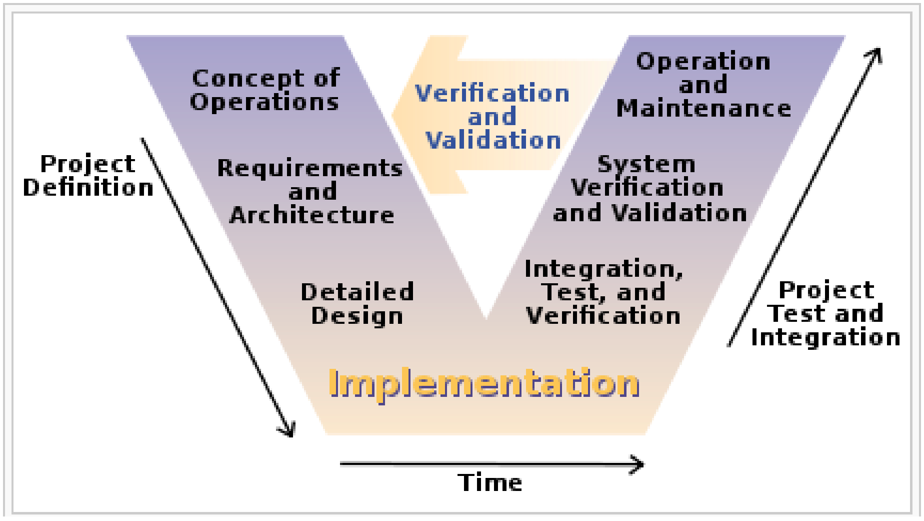
First, requirements needed to be extensively captured and analyzed before the work of development should commence. So, we would develop enormous spreadsheets of requirements, spending weeks on making sure that they represented what "the customer" wanted. The objective was to get the customer's signature, and then 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 available for weeks or months yet.)

Only AFTER extensive requirements, analysis, and design would coding take place (implementation). There was furthermore 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.


Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

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. And they would 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 (***Glass et al, but not Standish). This led to the perception of a “software crisis.” (It should also be noted that many large systems were effectively constructed and operated, and that there are reasonable criticisms of the concept of a “software crisis” (French author)).

It should be noted that 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. 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 XXXX and developed the Agile Manifesto, 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.

The Agile models for developing software were a fit with the rise of Cloud and Web-scale IT. As new customer-facing sites like Amazon, Netflix, Etsy, and Facebook scaled to massive proportions, it become increasingly clear that waterfall approaches were a failure. 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.

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 full understood in any event.) This has paid off in many cases.

A continuous delivery pipeline

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). At this writing, some very characteristic toolsets might include:

Stack 1	**Stack 2**	**Stack 3**	**Stack 4**
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

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. Text editors and IDEs are out of scope for this book, as they are usually 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 an overview.

Consider the Lean Software challenge by Mary and Tom Poppendieck: (***cite)

How long would it take you to change one line of code?

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.

Achieving this goal is feasible but requires a different approach…

```
Applied in an IT setting, a number of practitioners have explored this approach and
encountered great success. Key initial milestones included
```

- The establishment of “test-driven development” as a key best practice in creating software
- Duvall’s book “Continuous Integration”
- Allspaw & X’s seminal “10 Deploys a Day” presentation describing at Etsy
- Humble & Farley’s “Continuous Delivery”

There is a great deal written on the topic of DevOps currently. Some of the most critical practices can be seen by briefly viewing Jez Humble’s Chapter 1 headings for Continuous Delivery:

- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Build Quality In
- Done Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement (***cite)

Humble’s book is recommended unreservedly.

Let’s go into a little detail on the essential Agile practices.

- Test driven development
- Ongoing refactoring
- Continuous integration

- Continuous deployment

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, 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.

I’m sure we can find a million references on this

The requirement calls for work to be performed.

An 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 requirement leads to the artifact, and the artifact leads to the commit, it makes sense to associate the requirement with a fork 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.

Test-driven development and refactoring

Test-driven development. (Sidebar)

As Martin Fowler says,

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. (***http://martinfowler.com/books/meszaros.html)

In test-driven development, the idea essentially is writing 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. Thus:

1. Given a “user story” (I.e requirement), 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 indepth 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 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
  }
}
```

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

1. creates an instance of the class H1Class, based on a string “TestWebMessage”, and
2. confirms 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 programming environment 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. For example, it’s reported that Google runs 100 million automated tests daily. (cite). It has been successfully employed also in hardware development. (cite)

- Pyramid vs cupcake http://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern?utm_campaign=software-testing&utm_medium=social&utm_source=twitter

Refactoring

Test-driven development enables the next major practice, that of refactoring. As defined by Martin Fowler,

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 "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. (cite)*

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 project forwards. 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 Duvall in his book of the same name.

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?

These problems have driven the evolution of software configuration management for decades. 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 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.

And so it goes. These practices arose in response to the phenomenon of “merge hell” that was seen in older development approaches. 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 (blog re: DVCS hype)
- 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.

The idea that "Crucially, if the build fails, the development team stops whatever they are doing and fixes the problem immediately" (***humble) has been argued against at http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html.

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," anyways? 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.

Conclusion

Looking forward: closing the loop via operations management

Discussion questions

Research & practice

Further reading

- Fowler, Refactoring
- McConnell, Code Complete.
- Humble & Farley, Continuous Delivery
- Kim et all, Phoenix Project
- Continuous Integration

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.

should we do section I readings/questions?

Section II: Collaboration

Section II introduction

Scenario

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. With 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 it's not running right and people can't get to it.

Sidebar: Systems theory and idea of feedback

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.

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.

Chapter 4: Product Management

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. User experience and behavior-driven design are key factors here.

Furthermore, while we covered Agile principles and practices in some detail last chapter, 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 4 outline

- Lean introduction.
  - Lean manufacturing vs Lean product development
  - Applicability to IT

Before we dive into the chapters, however, let’s briefly cover one of the most important industrial forces affecting IT today: the Lean movement. *etc*

Sidebar: Introducing Don Reinertsen.

Assumption: Product strategy was largely tacit in Section I. Need more formalized approaches to defining, communicating, executing.

- Product vs project management
- Lean Startup
  - Minimum Viable Product etc
- LeanUX and Continuous Design.
- Behavior-driven development.
- The role of the product owner.
- Queuing?? (Or in chapter 8?)
- Importance of small batch sizes
- Product development as creation of information

Sidebar: Introducing Sussna.

see http://www.aha.io/ - lab?

Chapter 4 learning objectives

- Define and distinguish product versus project management
- Define and distinguish Lean manufacturing versus Lean product development
- Describe key principles of user experience (UX) and behavior-driven development
- Describe relationship between small batch sizes, limited work in process and fast feedback

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

- Reinertsen, Principles of Product Development Flow
- http://blog.fastmonkeys.com/2014/06/18/minimum-viable-product-your-ultimate-guide-to-mvp-great-examples/

Chapter 5: Work Management

Introduction to chapter 5

Instructor's note

"Work management" is an unusual title, deliberately chosen to distinguish the emergence of tracking and ticketing systems at earlier stages of organizational growth. At this point, a fully realized process framework may not be 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.

You're a team. You need to get work done. 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 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 also may become aware of the idea of "ticketing," if you have people with previous service desk experience. How this relates to your Kanban approach is also a question.

Sidebar: Introducing Anderson and Sutherland

Chapter 5 outline

- Kanban
  - Origins
  - Practices
  - Visualize
  - Limit WIP
  - To automate or no?
- Scrum
  - Origins of Scrum
  - Roles and practices
  - Relationship to product owner.
- Relationship of Scrum to Kanban
- Basic help desk ticketing
- A look ahead to more formalized process architectures

Chapter 5 learning objectives

- Define Kanban and identify its key practices
- Define Scrum and identify its key practices
- Compare and contrast Scrum and Kanban
- Describe basic help desk ticketing.

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

- Burrows

- Anderson
- Sutherland

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. Quesions 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" (vs. product mgmt)
- Kinds of work (where is that original concept)

  - interrupt-driven
  - etc
- OODA & situational awareness
  - Event management and monitoring
- Classic model (24x7, oncall, etc)
  - Mention ITIL (more depth in chapter 8)
- Impact & dependency

Sidebar: Configuration Management and the CMDB

- State & stability (Burgess?)
- Control theory?
- Web-scale IT (first pass)
- Antifragility

Chapter 6 learning objectives

- Distinguish kinds of work, esp operational vs development
- Describe operational feedback into product design & the kinds of concerns it raises
- Describe impact and dependency analysis and why it is important

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

- Allspaw
- Limoncelli

Section III: Coordination

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.

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. 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. Welcome to IT management, circa 2015.

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 We want to get bigger stuff done. Some argue that we don’t need projects at all any more, and it’s true that more and more we think in terms of **products** where we used to think of **projects**. But there still is a need to orchestrate the work; even Google and Facebook use project managers. We also have to understand what people are doing on a daily basis. 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 need certain resources and services delivered predictably. Our increasingly specialized organization needs some specialization of processes, and without some kind of coordination we run the risk of two teams trying to change the same thing at the same time. We also become specifically concerned with the process by which process is managed, i.e., continuous improvement.

Chapter 7: Culture and Organization

Introduction to chapter 7

Complexity responds to competence, not authority. Steve Denning

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 here.

Sidebar: Introducing Rother.

Chapter 7 outline

- What is organizational culture?
    - Extensively researched & written about, but less so for IT
    - Examples from Rother, Toyota Kata (good examples for answering "what is culture")
    - Taylor & cultural antipatterns (including the *you just don’t get X* antipattern)
    - Culture != *collaborate & be nice*

Sidebar: Introducing Narayan and Mintzberg.

- Organizational fundamentals
  - Line vs staff
  - Functional vs product-oriented example
  - Recent Dan Jones HBR article on Lean organization & functional expertise
  - T-shaped people as essential to product teams (Reinertsen principle)

Sidebar: Spotify overview

Chapter 7 learning objectives

- understand various concepts of culture
- identify key cultural themes from Toyota Kata
- distinguish between functional versus product organization
- *something with Mintzberg and/or Narayan*

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Chapter 8: Project & Resource Management

Project management is traditionally defined as the execution of a given scope of work within constraints of time and budget. As we have seen in our discussions of product management, in implementing truly new IT-based products, estimating time and budget can be difficult because the necessary information is not available. In fact, creating it is the actual work of the "project". Therefore, in the new Agile world, there is some uncertainty as to the role and even need for traditional project management.

However, at a practical level, project managers remain important roles and are employed at both "unicorn" and "horse" companies as overall orchestrators of complex initiatives. Financial planning has not gone away and without project management, financial planning meets difficult challenges. The very concept of having a vision becomes problematic without some ability to plan.

Because of this, this chapter also covers IT financial management at a very introductory level.

Introduction to chapter 8

Chapter 8 outline

- Introduction to project management
  - PMBOK etc (also covered in Appendix)
  - Scrum and XP (Scrum introduced in 5, need to figure out correct allocation)
  - Notice that Scrum originated from Sutherland’s study of product management, not project management
- Is project management still needed in a product-centric world?
  - yes. Google & FB both use it.
- Project management antipatterns

Sidebar: Introducing Scaled Agile Framework (Leffingwell)

- The estimation controversy (sidebar?)
- The future of project management
- Resource management (yes, it’s a terrible name) *get permission for Dilbert cartoon "I am not a resource"*
  - SFIA framework
  - Approaches to time tracking
- Supply and demand.

  - IT talent
  - DSE - alternate view to PBR]
  - 2 dimensions of demand management
  - More to come in portfolio mgmt
  - Last responsible moment
  - Ops theory - process, project, execution, resource & scheduling constraints
- Introduction to IT financial management

Chapter 8 learning objectives

- Identify the basic practices of project management
- Identify the basic concerns of IT resource management
- Identify the basic concerns of IT financial management

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Chapter 9: Process Management

Introduction to chapter 9

No matter what your organization structure, you need process. Toyota considers a clear process vision, or "target condition," to be the most fundamental objective in improving operations. Process, in terms of "business process," has been a topic of study and professional education for many years. Designing processes, improving them, and using them to improve overall performance is an ongoing activity in most if not all organizations.

However, some less than effective things have been done in the name of "process." Both ITIL and the Capability Maturity Model-Integrated claim to be "process"-oriented, but their results have been questioned. In particular, there is risk in allowing processes to proliferate - the organization can only tolerate a certain number of processes before people start to become confused and demoralized by all the "bureaucracy." Organzational improvement approaches that encourage the formation of processes and associated metrics at all levels may therefore not be an effective model.

Nevertheless, process remains an important tool in the toolkit for organization design.

Chapter 9 overview

Sidebar: Introducing ITIL.

Placeholder for chapter 8 introduction.

- Process management basics
  - BPM community
  - Process as a Mintzberg liaison mechanism

Sidebar: Can microservices eliminate most or all need for cross-team communication & liaison mechanisms?

- Function vs process
- Practice vs process
- process vs execution (execution includes concern for resource)
- "Process" at Toyota (Rother)

Sidebar: Lessons from manufacturing, operations research, etc

Controversy of application to IT

Will this have been foreshadowed?

- Self-organizing (kanban) processes vs. formalized process frameworks.
- IT "process" frameworks (ITIL etc - also in appendix)
- Process in product-centric organizations
- Process antipatterns (should I do more with antipatterns in each chapter?)
  - Process proliferation
  - Process without execution model (demand management failure)
- SLAs and the service catalog

Sidebar: Quality and Continuous Improvement

Chapter 9 learning objectives

- Describe a business process and how it crosses functions
- Describe the relationship between business process and business performance
- Describe the origins, purpose, and scope of ITIL and CMMI
- Identify the problems of process proliferation with respect to execution and demand

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Section IV: Maturation

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.

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: Architecture and Governance We need better orientation on how the big picture fits together. We have major players and forces around us (vendors, regulators, security threats). Some of the investments we have made in technology platforms are getting old. Switching to a new platform is looking to be a "bet the company" big bet. We also have technical debt. We need to reduce redundancy and we **do** need to seek some economies of scale.

Chapter 11: Portfolio and Analytics We need to define our investment strategy based on a sound understanding of both business needs and technology limitations. We need to measure this massive IT estate and understand it as an overall dynamic system, without falling into the trap of metrics for the sake of metrics.

Chapter 12: Complexity, chaos, and the road ahead No matter how we try, stuff happens, and it's getting harder to cope with as the systems get bigger and more complex. Where will this all lead?

Chapter 10: Architecture and governance

Introduction to chapter 10

Architecture and governance are two topics that become more important as a company scales up and ages. Decisions that might have been made quickly and casually by a product or project team become more difficult, due to increasingly challenging structural factors both internal and external. Decisions made years ago come back to haunt current strategies with a vengeance. With scale, management needs higher level technical abstractions in order to manage. Architecture provides these, but doing so is difficult and controversial. What happens when management decides to set direction using these same abstractions, and architects find themselves now enforcing what had originally started as mere 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 become corrupted by business realities.

Security threats increase proportionally to the company's size. The talent and persistence of these adversaries is remarkable. Another set of antagonists 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? Data protection, retention, ?

- Architecture
    - Also important in next chapter, which can be understood as "advanced architecture"

Sidebar: Introducing Open Group and OMG.

Chapter 10 outline

- Defining architecture
    - Enteprise
    - Business
    - Data

    - Solutions
    - Technical
- Architecture processes and practices
- IT governance, risk, compliance

Sidebar: Introducing ISACA

- Defining governance
- The role of internal audit

ask Patrick Stachenko to write sidebar on how audit moved into IT

- Governance, Risk & Compliance as a general space
- "Rogue" IT

Use of the word "architecture" in information technology

"Architecture" is of course a word usually associated with physical construction: buildings, landscapes, etc. It was appropriated by systems engineers at IBM in 1959 (https://en.wikipedia.org/wiki/Computer_architecture) 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.

Chapter 10 learning objectives

- Identify the basic types of IT architecture
- Describe basic architectural practices
- Characterize some typical challenges for architects

- Define governance vs. management
- Identify common regulatory compliance issues
- Discuss basic security concerns and practices

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Chapter 11: Portfolio and Analytics

Collaborator's note I am debating whether the chapters should be

- 10. Architecture and Analytics
- 11. Portfolio and Governance

Introduction to chapter 11

Portfolio management in terms of information technology means looking at long-lived IT investments in terms of their overall cost, risk, and benefit to the corporation. This can and should be done regardless of whether the IT investment is external or internal facing. Applications and services are the most useful portrfolio constructs, although assets, technology products, and even projects also can figure into longer-horizon value management. Enterprise architecture benefits from a tight alignment with IT portfolio management, as well.

Analytics and metrics are a related topic for IT management. There are many ways that metrics can be used and misused. A clear understanding of organizational goals is essential to any metrics strategy. Analytics similarly requre an overall framework of continuous improvement so that their insights lead to real actions and value.

Chapter 11 outline

- the IT lifecycles
    - Application service
    - Infrastructure service
    - Asset (including physical estate)
    - Technology Product
- IT analytics
    - grounded in continuous improvement
    - Portfolio and process data
    - The IT data warehouse
    - Looking ahead: Big Data, machine learning, text analytics

Sidebar: IT analytics experts Magennis, Cantor, etc

Chapter 11 learning objectives

- Identify the major IT lifecycles and their interactions
- Identify various use cases for IT analytics *

Chapter 11 content to be moved to new file 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 inordinate resource, and yet managing the IT organization at scale is nearly impossible without them.. [*didn’t want to discuss data per se in the introduction*]

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Lean Enterprise, Humble

Chapter 12: Complexity, chaos and the road ahead

Introduction to chapter 12

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?

Chapter 12 outline

Sidebar: Introducing Burgess.

- complexity (w/r/t Burgess)
- stability, state, etc
- complex system failures (Allspaw sidebar?)
- Cynefin
- Postscript: Thought experiment on complete mainstreaming of IT

Sidebar: Introducing Snowden & Taleb.

Learning objectives

- Identify current approaches to understanding complex systems and their failure modes
- Describe Snowden’s Cynefin and its major elements

Conclusion

Conclusion placeholder

Discussion questions

Research & practice

Further reading

Appendices & back matter

placeholder for appendices

- Frameworks overvew
- Architectural views (??)
- Towards a theory of IT management

References

- [Alliance2001]: Alliance, A. (2001), "Agile Manifesto and Principles", http://agilemanifesto.org/principles.html
- [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
- [Anderson2010]: Anderson, D. J. (2010), *Kanban: Successful Evolutionary Change for your Technology Business*, Blue Hole Press, Sequim, WA
- [Bell2010]: Bell, S. C. & Orzen, M. A. (2010), *Lean IT*, CRC Press, Boca Raton, Florida
- [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
- [Betz2015]: Betz, C.T. (2015), "Calavera project", Github, https://github.com/CharlesTBetz/Calavera
- [Bouwman]: Bouwman, J.-J. & Heistek, M. (), "ITIL and DevOps at war in the enterprise,"" https://www.youtube.com/watch?v=_dDsdbkSgOc, *DevOpsDays*.
- [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
- [Carroll2013]: Carroll, I. (2013), "Various", http://itopskanban.wordpress.com/before/
- [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/
- [Fowler2003]: Fowler, M. (2003), *Patterns of enterprise application architecture*, Addison-Wesley, Boston
- [Fowler1997]: Fowler, M. (1997), *Analysis patterns : reusable object models*, Addison Wesley, Menlo Park, Calif.
- [Gamma1995]: Gamma, E. (1995), *Design patterns : elements of reusable object-oriented software*, Addison-Wesley, Reading, Mass.
- [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
- [Hay2006]: Hay, D. C. (2006), *Data model patterns : a metadata map*, Morgan Kaufmann ; Oxford : Elsevier Science [distributor], San Francisco, Calif.
- [Hay1996]: Hay, D. C. (1996), *Data model patterns : conventions of thought*, Dorset House Pub., New York
- [Hohpe2003]: Hohpe, G. & Woolf, B. (2003), *Enterprise integration patterns : designing, building, and deploying messaging solutions*, Addison-Wesley, Boston
- [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
- [Kim2013]: Kim, G.; Behr, K. & Spafford, G. (2013), *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, IT Revolution Press

- [Kniberg2011]: Kniberg, H.; Beck, K. & Keppler, K. (2011), *Lean from the trenches : managing large-scale projects with Kanban*, Pragmatic Bookshelf, Dallas, Tex.
- [Krafcik1988]:Krafcik, J. (1988),"Triumph of the lean production system", *Sloan Management Review* 30(1), 41-52.
- [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
- [Minick2012]: Minick, E. (2012), "A DevOps Toolchain: There and back again", Slideshare.net, http://www.slideshare.net/Urbancode/building-devops-toolchain
- [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.
- [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
- [Reinertsen2009]: Reinertsen, D. G. (2009), *The principles of product development flow: second generation lean product development*, Celeritas, Redondo Beach, Calif.

- [Reinertsen1997]: Reinertsen, D. G. (1997), *Managing the design factory: a product developer's toolkit*, Free Press, New York ; London
- [Ries2011]: Ries, E. (2011), *The lean startup : how today's entrepreneurs use continuous innovation to create radically successful businesses*, Crown Business, New York
- [Scotland2010]: Scotland, K. (2010), "Defining the Last Responsible Moment", http://availagility.co.uk/2010/04/06/defining-the-last-responsible-moment
- [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/
- [Silverston2008]: Silverston, L. (2008), *The data model resource book Vol 3: Universal patterns for data modeling*, Wiley, Indianapolis, Ind.
- [Thompson2014]: Thompson, L. (2014), "Hitchhikers Guide to OpenStack Toolchains", https://www.openstack.org/assets/presentation-media/Hitchhikers-Guide-to-OpenStack-Toolchains.pdf
- [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

1. I admit my bias is US-centric. There is a European discipline called "informatics," that may be closer to what is needed.

2. 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.

Last updated 2015-08-06 11:22:43 CDT