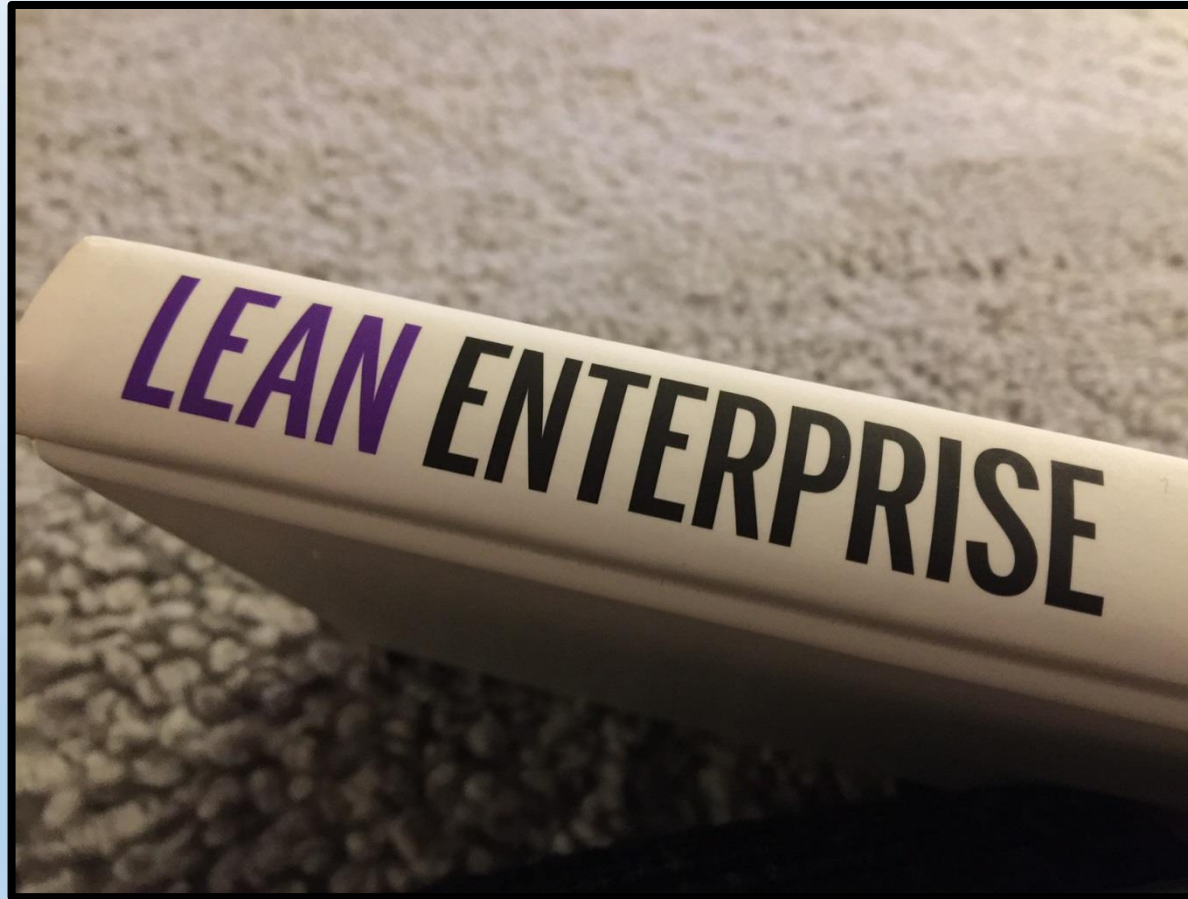



Executive Summary



Top Quotes from the book



DEVOPS



Engineers building new features should be able to push code changes live themselves, following peer review, except in the case of high-risk changes. However, they *must* be available when their changes go live so they can support them. Many new code changes (particularly high-risk ones) should be launched “dark” (as described in Chapter 8) and either switched off in production or made part of an A/B test.



Some people describe this model as “no-ops,”⁸ since (if successful) we drastically reduce the amount of reactive support work that operations staff must perform. Indeed teams running all their services in the public cloud can take this model to its logical conclusion where product teams have complete control over—and responsibility for—building, deploying, and running services over their entire lifecycle (a model pioneered at scale by Netflix). This has lead to a great deal of resistance from operations folks who are concerned about losing their jobs. The “no-ops” label is clearly provocative, and we find it problematic; in the model we describe, demand for operations skills is in fact increased, because delivery teams must take responsibility for operating their own services. Many IT staff will move into the teams that build, evolve, operate, and support the organization’s products and services. It is true that traditional operations people will have to go through a period of intense learning and

YES!!

It must be recognized and accepted that this will be scary for many people. Support and training must be provided to help those who wish to make the transition. It must be made clear that the model we describe is not intended to make people redundant—but everyone needs to be willing to learn and change

All changes to any system—or the environments it runs in—should be made through version control and then promoted via the deployment pipeline. That includes not just source and test code but also database migrations and deployment and provisioning scripts, as well as changes to server, networking, and infrastructure configurations.

IT SECURITY

To succeed, IT organizations must take one of the two paths: either outsource to external suppliers of infrastructure or platform as a service (IaaS or PaaS), or build and evolve their own.

While moving to external cloud suppliers carries different risks compared to managing infrastructure in-house, many of the reasons commonly provided for creating a “private cloud” do not stand up to scrutiny. Leaders should treat objections citing cost and data security with skepticism: is it reasonable to suppose your company’s information security team will do a better job than Amazon, Microsoft, or Google, or that your organization will be able to procure cheaper hardware?

Given that break-ins into corporate networks are now routine (and sometimes state-sponsored), the idea that data is somehow safer behind the corporate firewall is absurd. The only way to effectively secure data is strong encryption combined with rigorous hygiene around key management and access controls.

To meet compliance and reduce security risks, many organizations now include information security specialists as members of cross-functional product teams. Their role is to help the team identify what are the possible security threats and what level of controls will be required to reduce them to an acceptable level. They are consulted from the beginning and are engaged in all aspects of product delivery:

- Contributing to design for privacy and security
- Developing automated security tests that can be included in the deployment pipeline

Amazon.⁹ Many countries are now updating their regulations to explicitly allow for data to be stored in infrastructure that is externally managed.

There are two good reasons to be cautious about public clouds. The first risk is

Architect the new software to run on a PaaS

Work with operations to drive the design of the software hand in hand with the platform as a service, as we describe in Chapter 14. If the operations team is not ready to do this, work with them to ensure that the system doesn’t drive up the complexity of the existing operational environment.

Any cloud implementation project not resulting in engineers being able to self-service environments or deployments instantly on demand using an API must be considered a failure. The only criterion for the success of a private cloud implementation should be a substantial increase in overall IT performance

ity]. He calls it the ‘wouldn’t it be horrible if...’ approach. In this framework, IT security specialists imagine a particularly catastrophic event occurring. Regardless of its likelihood, it must be avoided at all costs. Tippet observes: ‘since every area has a “wouldn’t it be horrible if...” all things need to be done. There is no sense of prioritization.’⁵

COMPANY CULTURE

Responsibility

Each individual is responsible for the activities, tasks, and decisions they make in their day-to-day work and for how those decisions affect the overall ability to deliver value to stakeholders.

work on safety culture shows that *no* process or control can compensate for an environment in which people do not care about customer and organizational outcomes. Instead of creating controls to compensate for pathological cultures,

People are trusted to make the best decisions in their context, but are accountable for those decisions—in terms of both the achieved outcomes and knowing when it is appropriate to involve others.

the solution is to create a culture in which people take responsibility for the consequences of their actions—in particular, customer outcomes.

started by civil servants, not
An innovation culture is created by harnessing people's need for mastery, autonomy, and purpose—and making sure people are deeply committed to the organization's purpose and the users they serve.

resources. However, in high-performance organizations, projects and requirements are not tossed over the wall to IT to build. Rather, engineers, designers, testers, operations staff, and product managers work in partnership on creating high-value outcomes for customers, users, and the organization as a whole. Furthermore, these decisions—made locally by teams—take into account the wider strategic goals of the organization.

IT MANAGEMENT

teams on a day-to-day basis. Enterprises often waste a great deal of time on unnecessary, disruptive reorganizations—when they would do better simply by having people who work on the same product or service sit all in the same room (or, for larger products, on the same floor).

to the target condition. In the Improvement Kata, people doing the work strive to achieve the target condition by performing a series of experiments, not by following a plan.

The great planning fallacy, evident in the centralized budget process, is that if we develop a detailed upfront financial plan for the upcoming year, it will simply happen—if we stick to the plan. The effort to develop these kinds of plans is a waste of time and resources, because product development is as much about discovery as it is about execution. Costs will change, new opportunities will arise, and some planned work will turn out not to generate the desired outcomes. In today's world of globalization, rapid technology growth, and increasing unpredictability it is foolish to think that accurate, precise plans are achievable or even desirable.

A better approach is to set high-level long-term goals, carefully manage the more predictable near future, and constantly adjust our shorter-range plans to get closer to our targets. We can adopt this approach by implementing strategy

their team and rotate through them. The TPS also removes the visible trappings and privileges of management. Nobody wore a tie at the NUMMI plant—not even contractors—to emphasize the fact that everybody was part of the same team. Managers did not receive perks accorded to them at other GM plants, such as a separate cafeteria and car park.

When you practice the Improvement Kata, process improvement becomes planned work, similar to building product increments. The key is that we don't plan *how* we will achieve the target condition, nor do we create epics, features, stories, or tasks. Rather, the team works this out through experimentation over the course of an iteration.

In enterprises, one indicator of too much WIP is the number of people assigned to more than one project. This pernicious practice inevitably leads to longer lead times and lower quality due to constant context switching. Instead of assigning people to multiple projects, have a centralized team that can provide extra specialist support to teams on demand, but do not assign these people to any teams and carefully monitor their utilization to keep it well below 100%.⁸

punished for failing to meet targets or metrics, one of the surest ways to start manipulating work and information to look like they are meeting the targets. As FutureSmart's experience shows, having good real-time metrics is a better approach than relying on scrums, or scrums of scrums, or Project Management Office reporting meetings to discover what is going on.

This is significantly different from how work is planned and estimated in large projects that often create detailed functional and architectural epics which must be broken down into smaller and smaller pieces, analyzed in detail, estimated, and placed into a prioritized backlog *before* they are accepted into development.

(COTS). Whenever you hear of a new IT project starting up with a large budget, teams of tens or hundreds of people, and a timeline of many months before something actually gets shipped, you can expect the project will go over time and budget and not deliver the expected value.

OTHERS

IT operations—a department within the IT department and perhaps the ultimate cost center—experiences the consequences of these decisions on a daily basis. In particular, the integrated systems they must keep running are incredibly complex and crufty, built up over years, and often fragile, so they tend to avoid changing them. Since stability is their first priority, IT operations has developed a reputation as the department that says “no”—an entirely rational response to the problems they face.

- Test automation can become a maintenance nightmare if automated test suites are not effectively curated. A small number of tests that run fast and reliably detect bugs is better than a large number of tests that are flaky or constantly broken and which developers do not care about.

Our experience is that standardization on a particular toolchain or technology stack is neither necessary nor sufficient for achieving enterprise architecture goals such as enabling teams to respond rapidly to changing requirements, creating

- Only spend time and effort on test automation for products or features once they have been validated. Test automation for experiments is wasteful.

This problem is exacerbated by the typical project model through which IT projects are funded and managed. The work created in IT projects is typically handed over (or thrown over) to IT operations to run, so the people managing the projects have little incentive to think about the long-term consequences of their design decisions—and large incentives to ship as much functionality as

Continuous integration is the practice of working in small batches and using automated tests to detect and reject changes that introduce a regression. It is, in our opinion, the most important technical practice in the agile canon, and it forms the foundation of continuous delivery, for which we require in addition that each change keeps the code on trunk releasable. However, that can be

devices, as shown in Figure 6-1. Moving away from branch-based development to trunk-based development was also necessary to implement continuous integration. Thus the team decided to create a single, modular platform that could

“Trust, but verify”⁷ is a concept that is gaining acceptance in GRC circles. Instead of preventing teams from accessing environments and hardware so they can’t do anything bad, we trust people to do the right thing and give the team access and control on the systems and hardware they need to use daily. We then verify the team is not abusing their authority by developing good monitoring and frequent review processes to ensure the established boundaries are observed and there is complete visibility and transparency built into the team’s work.

ring. We all manage risks daily, at work, home, and play. As it is impossible to eliminate every risk, the question to be answered in managing risk is, “Which risks are you willing to live with?” As you take steps to mitigate risk in one area, you inevitably introduce more risk in another area. A classic example of this is restricting development team access to hardware and forcing them to rely on a separate centralized infrastructure team to set up access and environments for testing or experiments. This may be effective for the server support team’s goal of reducing the risk of instability within systems, but it increases the risk of delayed delivery as teams have to submit requests to other teams and wait for them to be fulfilled.

CHANGE CONTROL

Continuous Delivery and Change Control

Many enterprises have traditionally used change advisory boards or similar change control systems as a way to reduce the risk of changes to production environments. However, the *2014 State of Devops Report*,¹¹ which surveyed over 9,000 individuals across many industries, discovered that approval processes external to development teams do little to improve the stability of services (measured in terms of time to restore service and percentage of failed changes), while acting as a significant drag on throughput (measured in terms of lead time for changes and change frequency). The survey compared external change approval processes with peer-review mechanisms such as pair programming or the use of pull requests. Statistical analysis revealed that when engineering teams held themselves accountable for the quality of their code through peer review, lead times and release frequency improved considerably with negligible impact on system stability. Further data from the report, which supports the use of the techniques discussed in this chapter, is presented in Chapter 14.

Deployment is the installation of a given version of a piece of software to a given environment. The decision to perform a deployment—including to production—should be a purely technical one. *Release* is the process of making a feature, or a set of features, available to customers. Release should be a purely business decision.

REGULATIONS AT AMAZON

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols—doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.