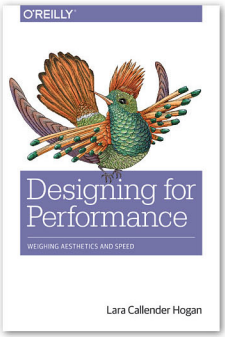
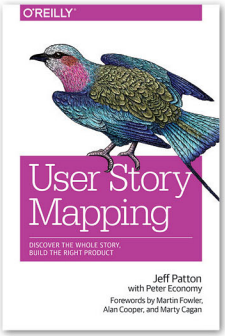
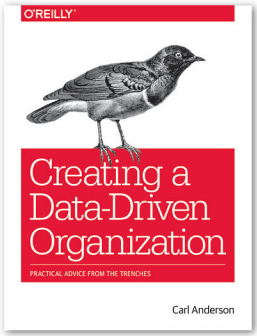
O'REILLY®

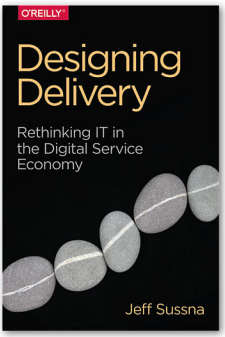
Building an Optimized Business

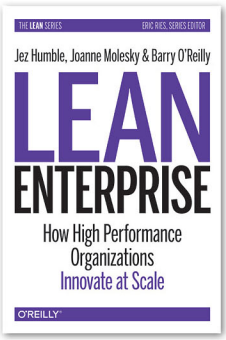
What Business Leaders Need to Know about Operating at Speed and Scale

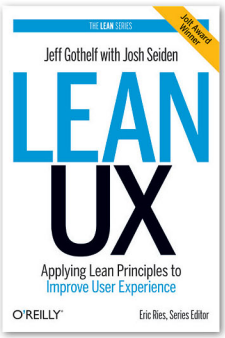


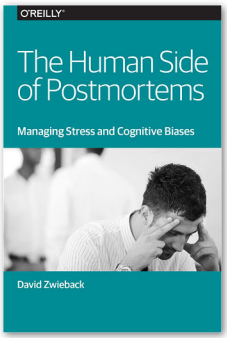


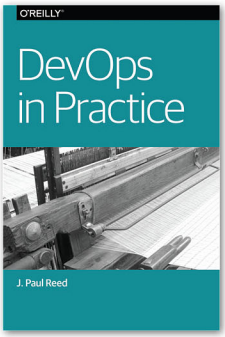












What Business Leaders Need to Know About Operating at Speed and Scale

The post-industrial shift that sociologist Daniel Bell foresaw more than 40 years ago is here: Disruption is driving business and changing entire industries. IT isn't just moving beyond its supporting role in business operations, it's becoming inseparable from it. Every business is becoming a digital business and "innovate or die" is the business mantra of the day.

Companies now realize the need to empower their employees' creativity and decision-making abilities, and they're looking to transform organizational structures and communications tools to unleash internal innovation and collaboration. The question is: how?

To help you navigate this changing landscape, O'Reilly offers a collection of standalone chapters from several of its published and forthcoming books. This sampler provides valuable information on DevOps, lean development, changing to a data-driven culture, and other related subjects.

For more information on current and forthcoming Velocity content, check out *http://www.oreilly.com/webops-perf/*.

—Courtney Nash, Strategic Content Lead, *courtney@oreilly.com*

The ebook includes excerpts from the following books:

Designing Delivery

Available in Early Release

Chapter 1. From Industrialism to Post-Industrialism

Lean Enterprise

Available here

Chapter 6. Deploy Continuous Improvement

DevOps in Practice

Available here

Chapter 2. Nordstrom

User Story Mapping

Available here

Chapter 3. Plan to Learn Faster

Lean Enterprise

Available here

Chapter 9. Take an Experimental Approach to Product Development

Lean UX

Available here

Chapter 7. Integrating Lean UX and Agile

Designing for Performance

Available here

Chapter 8. Changing Culture at Your Organization

Creating a Data-Driven Organization

Avalable in Early Release

Chapter 1. What Do We Mean by Data-Driven?

Human Side of Postmortems

Available here

Lean Enterprise

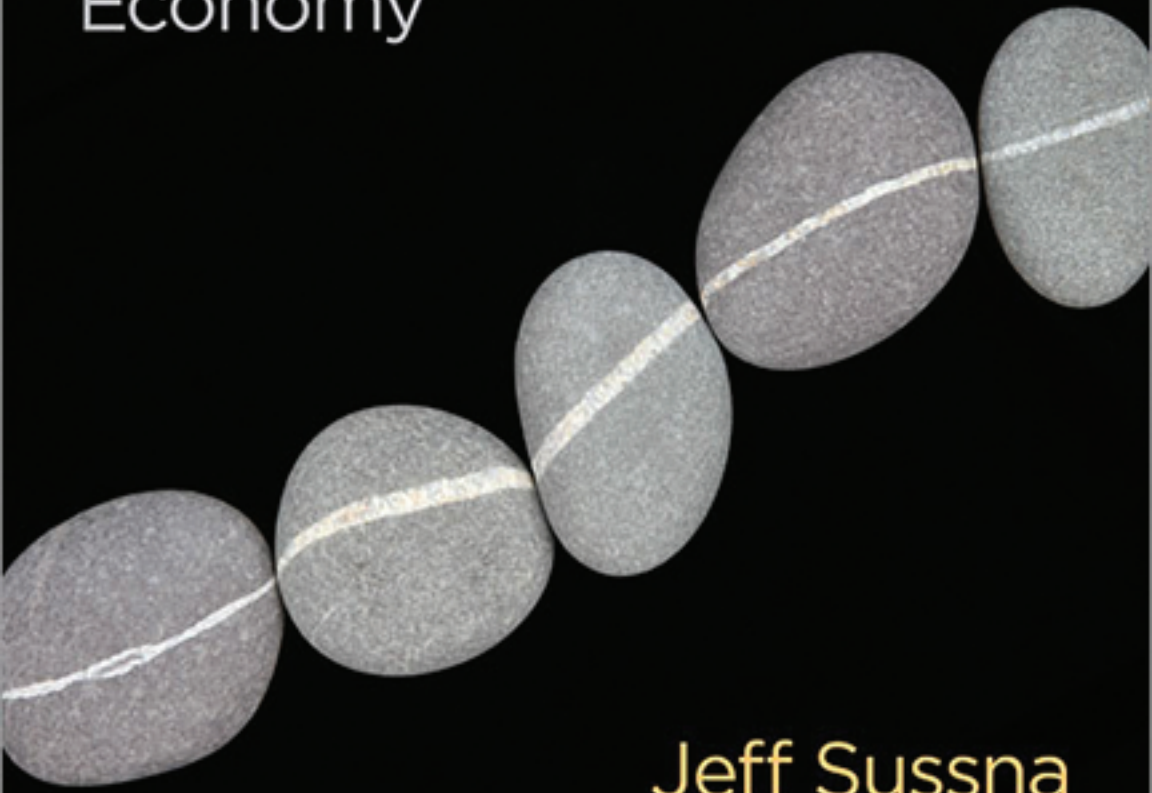
Available here

Chapter 15. Start Where You Are

O'REILLY®

Designing Delivery

Rethinking IT in the Digital Service Economy



Jeff Sussna

From Industrialism to Post-Industrialism

In 1973, Daniel Bell published a book called "The Coming of Post-Industrial Society". In it, he posited a seismic shift away from industrialism towards a new socio-economic structure which he named 'post-industrialism'. Bell identified four key transformations that he believed would characterize the emergence of post-industrial society:

- Service would replace products as the primary driver of economic activity
- Work would rely on knowledge and creativity rather than bureaucracy or manual labor
- Corporations, which had previously strived for stability and continuity, would discover change and innovation as their underlying purpose
- These three transformations would all depend on the pervasive infusion of computerization into business and daily life

If Bell's description of the transition from industrialism to post-industrialism sounds eerily familiar, it should. We are just now living through its fruition. Every day we hear proclamations touting the arrival of the service economy. Service sector employment has outstripped product sector employment throughout the developed world [1].

Companies are recognizing the importance of the customer experience. Drinking coffee has become as much about the bar and the barista as about the coffee itself. Owning a car has become as much about having it serviced as about driving it. New disciplines such as service design are emerging that use design techniques to improve customer satisfaction throughout the service experience.

1. *http://www.worldbank.org/depweb/beyond/beyondbw/begbw_09.pdf*

Disruption is driving hundred year-old, blue-chip companies out of business. Startups that rethink basic services like hotels and taxis are disrupting entire industries. *Innovate or die* is the business mantra of the day. Companies are realizing the need to empower their employees' creativity decision-making abilities, and are transforming their organizational structures and communications tools in order to unleash internal innovation and collaboration.

IT is moving beyond playing a supporting role in business operations; it's becoming inseparable from it, to the point where every business is becoming a digital business. One would expect a company that sells heating and ventilation systems to specialize in sheet aluminum and fluid dynamics. You couldn't think of a more 'physical-world' industry. Yet HVAC suppliers have begun enabling their thermostats with web access in order to generate data for analytics engines that automatically fine-tune heating and cooling cycles for their customers. As a result, they're having to augment their mechanical engineering expertise with skills in building and running large-scale distributed software systems.

From Products to Service

The Industrial Age focused on optimizing the production and selling of products. Interchangeable parts, assembly lines, and the division of labor enabled economies of scale. It became possible to manufacture millions of copies of the same object. Modern marketing evolved to convince people to buy the same things as each other. Consumerism brought into being a world where people evaluated their lives by what they had, rather than how they felt.

A product economy functions in terms of transactions. A sneaker company, for example, calculates how many units they think they can sell, at what price point, to whom. They create a marketing campaign to drive the desired demand. They link their production and distribution systems to that forecast.

The consumer, for their part, comes home from a run with sore feet and decides they need a new pair of running shoes. They go to the local athletic store and try on a few different kinds of shoes. At some point they make a decision and buy something, at which point the transaction is concluded.

In addition to being transaction-oriented, a product economy relies on a push marketing model. Companies use the Four P's (Production, Price, Promotion, and Place) to treat marketing like an "industrial production line that would automatically produce sales" [2]. According to this model, proper planning almost pre-destines

2. Wim Rampen, personal communication

the customer to drive to a certain store, try on certain shoes, and make a certain purchase.

The twentieth-century media model developed hand-in-hand with consumerist marketing. Broadcast television evolved as the perfect medium for companies try to convince consumers that they needed a particular product. The very words "broadcast" and "consumer" give away nature of the industrial production and marketing relationship.

A post-industrial economy shifts the focus from selling products to helping customers accomplish their goals through service. Whereas products take the form of tangible things that can be touched and owned, service happens through intangible experiences that unfold over time across multiple touchpoints. Consider the example of flying from one city to another. The experience begins when you purchase your ticket, either over the phone or via a website. It continues when you arrive at the airport, check your bags, get your boarding pass, find your gate, and wait for boarding to begin.

Only after you board does the flight actually begin. You buy a drink and watch a movie. Finally the plane lands; you still have to disembark and collect your luggage. The actual act of flying has consumed only a small part of the overall trip. In the process of that trip, you have interacted with ticket agents, baggage handlers, boarding agents, and flight attendants. You have interfaced with telephones, websites, airport signage, seating areas, and video terminals.

Unlike product sales, which generate transactions, service creates continuous relationships between providers and customers. People don't complain about United on Twitter because their flight was delayed. They complain because "as usual" their flight was delayed. Perhaps instead they remark on the fact that, for once, their flight wasn't delayed.

Service transforms the meaning of *value*. A product-centric perspective treats value as something to be poured into a product, then given to a customer in exchange for money. If I buy a pair of sneakers, then leave them in my closet and never wear them, I don't feel entitled to ask for my money back.

Service value only fully manifests when the customer uses the service. The customer "co-creates" value in concert with the service provider. The fact that an airline owns a fleet of airplanes, and sells you a ticket for attention seat on one of them, doesn't by itself do you any good. The value of the service can't be fully realized until you complete your flight. You and the airline, and its ticket agents, pilots, flight attendants, and baggage personnel, all have to work together in order

for the flight to be successful. The goals, mood, situation, and surrounding experiences, you bring with you all contribute to the success of the service experience.

Service changes the dynamic between vendor and consumer, and between marketer/salesperson and buyer. In order to help a customer accomplish a goal, you need to understand what their goals are, and what they bring with them to the experience. In order to do that, you need to be able to listen, understand, and empathize. Service changes marketing from "push" to "pull".

Marketers are beginning to adapt to this new model. They are recognizing that strategies like "content marketing" fail to provide sufficient visibility into the mind of the consumer. Some organizations are supplementing content with marketing applications. These applications flip the Four P's on their head, and give marketers meaningful customer insight through direct interaction.

Astute readers will notice the need for an even larger network of collaboration. The airline operates within an airport, which operates within a city. A successful trip therefore also needs help from security agents and road maintenance crews. High-quality services address the larger contexts within which they co-create value with customers.

Sun Country, for example, is a regional airline headquartered in Minneapolis. Those of us who live in Minneapolis know that Minnesota has two seasons: winter and road construction. Sun Country recognizes that road construction can cause driving delays. They post warnings on the home page of their website about construction-related delays on the routes leading to the airport.

Sun Country understands that, even though the roads around the airport are beyond their control, they can still impact the perceived quality of the service experience. If I arrive at the gate late and feeling harried, I'll have less patience for any mistakes on the part of the ticket agent. I'll more likely to find fault with the airline, regardless of who's truly at fault.

The Internet and social media are accelerating the transformation of the marketing and sales model by upending the customer-vendor power structure. Customers now have easy access to as much if not more information about service offerings and customer needs than the vendors themselves. Facebook and Twitter instantly amplify positive and negative service experiences.Customer support is being forced out onto public forums. Companies no longer control customer satisfaction discussion about their own products. Instead, they are becoming merely one voice among many.

From Discrete to Infused Experiences

It used to be relatively straightforward to know where in your life you were and what you were doing at any point in time. You were either at home or at work, or else driving between them. If you wanted to hang out with your friends, you went to the mall. If, on the other hand, you wanted to be alone, you went in your room and shut the door. If you wanted to surf the Internet, you sat down at a desk in front of a computer. If not, you went out for a walk or a drive.

Now, though, the parts of our lives are melding together and infusing each other. Do you go to the coffee shop to chat or to work? Do you use your phone to call your Mom, or to upload photos of your cat to Instagram, or to check your office email? Do you go to the library to look up hardcover books on shelves, or e-books on websites? Do you use your car for transportation, or as an incredibly complicated and expensive online music player? The answer to all of these questions is "yes".

Even within the digital domain, our daily activities, and the tools we use to accomplish them, are blending together. We use the same phones and laptops and cloud services to manage personal and business data. We check our Facebook accounts from work, and read our work email at the kitchen table. We use Twitter to maintain both friendships and professional networks.

Digital infusion has fully blossomed. The word "infusion" refers to the fact that computer systems are no longer separate from anything else we do. The digital realm is infusing the physical realm, like tea in hot water. Or, as Paolo Antonelli, Senior Curator of Architecture and Design for the Museum of Modern Art in New York, put it, "We live today not in the digital, not in the physical, but in the kind of minestrone of the two". [3]

We encounter fewer and fewer situations that are purely physical. When I go shopping for a new refrigerator, I'll likely read an online review of it on my smartphone at the same time that I'm looking at it on the showroom floor. IKEA has integrated its paper catalog with its mobile app. If I use my phone to take a picture of an item in the catalog, the app will bring up more detailed, interactive information about the item in question.

Digital infusion means that brick-and-mortar retailers like Sears and IKEA, which traditionally specialized in the in-store experience, now must also offer equally compelling online experiences. To make things even more challenging, customers expect seamless experiences across physical and virtual channels: stores, kiosks, web browsers, tablets, phones, cars, and so on. As a result, companies are

3. *https://twitter.com/lucarosati/status/495504935812075521*

having to expand their marketing, design, and technology expertise to bridge the physical and digital domains.

The digital realm has moved beyond an isolated, contained part of our lives to become the underlying substrate upon which we carry out all our activities and interactions. With the emergence of the Internet of Things, the digital realm is beginning to completely surround us. Our walls have connected thermostats. Our arms have connected watches. Our lawns have connected sprinkler systems. Our cars have connected dashboards.

In order to serve this newly infused world, companies need to undertake equally deep internal transformations. IT used to be a purely internal corporate function. IT might impact internal operations efficiency; from the consumer's perspective, though, it remained invisible, literally behind-the-scenes, like an automobile assembly line. The ease or pain with which employees shared marketing documents, or filed expense reports, or tracked vendor purchase orders, was of no concern to the customer.

Infusion breaks down the boundaries between internally facing systems of record and externally facing systems of engagement. The relational, continuous, collaborative nature of service means that internal company operations are inseparable from customer service. In a digitally infused business, therefore, IT becomes an integral part of the customer-facing service. In order for a customer to be able to upgrade a service subscription, for example, a public website may need to interact in real-time with a back-office ERP system. If that ERP system is slow, or incapable of providing important data back to the website, its failures will become visible to the customer.

The virtualization of experience dramatically raises the stakes for digital service quality. Quality becomes that much more important because people depend on digital services for their very ability to function. If I can't transfer money over the web from my savings account to my checking account, I might bounce my rent check. If the software in my thermostat has a bug, I can't warm up my house on a cold day. If my corporate ERP system goes down, my customers might not be able to log into their accounts.

From Complicated to Complex Systems

Digital infusion changes, not just the way we experience things, but also the way we organize, construct, and operate them. If the music player in your car isn't working, is it Honda's fault, or Pandora's? If you can't watch a video on Friday night, is it Netflix' fault, or Comcast's, or Tivo's? If you're a freelancer, do you work for

yourself, or your client, or the broker who got you the gig? Is your invoicing data managed by your accounting SaaS provider, or by the PaaS on top of which they run, or by the IaaS on top of which that PaaS runs? If you have a problem with something someone sold you, do you call their their customer support line, or do you just post your question or complaint on Facebook or Twitter?

Infusion breaks down familiar boundaries and structures. No longer can customers assume that Honda will transparently manage all of its vendors in order to deliver a working car, or be able to fix problems with any of its parts. Conversely, Honda can no longer assume it controls the communications channels with its customers.

This dissolving of boundaries impacts IT structures as well. If a customer can't log in to your website, is the problem caused by the web server, or the mainframe finance system that holds the customer record? The new requirement for interconnectivity between systems of record and systems of engagement complicates network and security architectures. So-called "rogue" or "shadow" IT, where business units procure cloud-based IT services without the participation of a centralized IT department, makes it harder to control, or even know, which data lives inside the corporate data center, and which lives in a cloud provider's data center.

Infusion forces homogeneous, hierarchical, contained systems to become heterogeneous, networked, fluid and open-ended. In other words, complicated systems become complex ones. People often use the words 'complicated' and 'complex' interchangeably. When applied to systems, however, they mean very different things, with very different implications for defining and achieving quality. We therefore need to understand the distinction between them.

COMPLICATED SYSTEMS

A complicated system may have many moving parts. A car contains something on the order of 30,000 individual parts. All 30,000 parts, though, don't directly interact with each other. The fuel system interacts with the engine, which interacts with the drivetrain. The fuel system consists of a fuel tank, a fuel pump, and a carburetor. The carburetor is made up of jets, float bowls, gaskets, and so on. (Sidebar: examples of complicated systems)

Complicated systems arrange their components into navigable, hierarchical structures that facilitate understanding and control. Very few of us can fix our own cars anymore. We can still, though, reasonably understand their overall structure. If our car has a flat tire, and the service technician tells us we need a new carburetor, we know enough to suspect that something fishy is going on.

The interactions within complicated systems don't dynamically change. The carburetor doesn't suddenly start directly interacting with the tires. Furthermore, complicated systems behave coherently as wholes. If you're driving your car, and you turn the steering wheel to the left, the entire car goes to the left. One of the doors doesn't decide to wander off in the opposite direction.

COMPLEX SYSTEMS

Complex systems, on the other hand, consist of large numbers of relatively simple components that have fluid relationships with many other components. Examples of complex systems include everything from ant colonies to companies to cities to economies. You can't really define an economy, for example, as a neat hierarchy, with the Chair of the Federal Reserve at the top, the Fortune 500 below that, and small businesses and sole proprietors at the bottom.

Instead, companies and individuals dynamically create and dissolve business relationships with each another on multiple levels. I hire a plumber. A large company buys a smaller one. An executive quits their position to found a startup competitor. Toyota buys parts from many different vendors. A battery manufacturer, on the other hand, may supply batteries to Toyota, Ford, and BMW.

Complex systems function more like an ongoing dance, with the dancers changing partners on the fly. Schools of fish and flocks of birds offer compelling illustrations of complexity. A bird flying within a flock can position itself next to any other bird within that flock. It can change positions at will. It decides where to fly next in concert with the other birds that happen to be near it at any given time.

EMERGENCE

Complex systems arise from non-linear interactions between their components. That's a fancy way of saying that the whole is greater than the sum of the parts. You can't capture the behavior of the flock by examining the behavior of the individual birds. The beautiful, fascinating, mysterious ebb and flow of the flock represents a property of complexity known as *emergence*. Emergent characteristics exist at the system level without any direct representation at the level of individual components.

Birds fly according to three simple rules:

1. Fly in the same general direction as your immediate neighbors
2. Fly toward the same general destination as your immediate neighbors

3. Don't fly too close to your immediate neighbors

A group of birds flying according to these rules will generate a pattern that we perceive as "flocking". There is nothing in the rules, though, that directly explains that pattern. One might even say that the "flock" really only exists in our minds.

A complex system like a flock of birds may display emergent, coherent patterns. Those patterns, however, result from the behavior of components making individual, independent decisions. Each bird within a flock decides for itself how to respond to any given situation. A car whose doors could wander off, and come back again, would need a much more flexible definition of structural coherence. Otherwise it would quickly fall apart.

This difference in structural coherence illustrates a critical difference between complicated and complex systems. Complicated systems rely on centralized control and hardwired organizational structures. As a result, they work very efficiently until they break. The fact that the parts of a car all hang together is good for streamlining and thus fuel efficiency. If a wheel falls off, though, the entire car comes to a grinding halt.

Complex systems, by contrast, are sloppy and prone to component failure, yet highly resilient. Their decentralized, fluid structure trades efficiency for resilience. A flock of birds that encounters a giant oak tree happily splits apart into pieces and flies around it. The flock then "glues" itself back together again on the other side. A few birds might unfortunately fly into the tree. The flock as a whole, though, is unharmed by its encounter with a large obstacle. By contrast, an airplane that tried to split itself into pieces and fly around a similar obstacle would fall to the ground and crash.

Emergence presents both challenges and opportunities to organizations trying to manage complex socio-technical systems. On the one hand, it requires tolerance for failure and apparent inefficiency. On the other hand, it offers a decentralized, responsive, and scalable approach to achieving success, whether defined as control, quality, competitiveness, or profitability. Organizational methodologies that leverage the power of emergence can help companies achieve strategic coherency without sacrificing tactical flexibility.

CASCADING FAILURES

At the same time that complex systems demonstrate resilience, though, they are also subject to the phenomenon of *cascading failure*. A cascading failure is one that occurs at a higher system level than an individual component. In a complex system,

failures can also result from the interactions between components. System-level failures can even happen while all the individual parts are operating correctly.

Contemporary industrial safety research explores this phenomenon. It might be possible, for example, to extend the maintenance schedule for an airplane part without violating the acceptable wear tolerance for that part. Taken together with similar changes elsewhere within the system, however, that extension might tip the system as a whole into an unsafe state.

The potential for cascading failures challenges complicated-systems approaches to planning and quality assurance. Reductionist techniques that break systems into their parts are insufficient for modeling complexity. In order to understand each component, and its potential to cause problems, one must also consider its relationships with other components.

SENSITIVITY TO HISTORY

Finally, complex systems exhibit what's known as *sensitivity to history*. Two similar systems with slightly different starting points may dramatically diverge from each other over time. This phenomenon is known as the "Butterfly Effect". The Butterfly Effect describes the imagined impact of a butterfly flapping its wings on weather patterns on the other side of the world. Were the butterfly in Singapore not have flown away from a flower at precisely the time that it did, so the parable goes, a hurricane might not have come into being in North Carolina.

In August of 2013 the Nasdaq trading systems went offline for the better part of a day. The reasons for the outage present a fascinating example of cascading failure coupled with sensitivity to history. The "root cause" of the outage was unusually high incoming traffic from external automated trading systems. The rapidly accelerating traffic triggered a fail-safe within Nasdaq's software systems that caused them to fail over to a backup system. That same traffic level triggered a bug in the backup systems that took them completely off line.

One might point the finger at the bug as the cause of the outage. Ironically, though, the problem started because of software doing exactly what it was supposed to do: fail over based on load. That failover was intended to function as a resilience mechanism. The outage also might never have happened had the morning's traffic profile been just a little bit different. Had it not peaked quite as high as it did, or accelerated quite as quickly, the bug might not have been exposed, and the failover might have worked perfectly. Alternatively, the failover logic might not have triggered at all, and the primary systems might have struggled successfully through the morning.

Emergence, cascading failure, and sensitivity to history conspire to make it infeasible to predict, model, or manage complex systems in the same ways as complicated systems. Trying to manage them too tightly, using traditional top-down command-and-control techniques, can backfire and turn resilient systems into brittle ones. The ability to survive, and even thrive, in the presence of failure is a hallmark of complex systems. Fires renew the health of forests. Attempts to prevent them often have the counterproductive effect of creating the conditions for catastrophic fires that destroy whole forests.

Instead, post-industrial organizations need to approach management with a new-found willingness to experiment. When prediction is infeasible, one must treat one's predictions as guesses. The only way to validate guesses is through experimentation. Just as complex systems are rife with failure, so too are the attempts to manage them. Experiments are as likely to return negative results as positive ones. Management for resilience requires a combination of curiosity, humility, and willingness to adapt that is that is unfamiliar and counterintuitive to the industrial managerial mindset.

REAL-WORLD COMPLEXITY

Complexity is more than just theoretically interesting. It increasingly presents itself in real-world business and technical scenarios. Employees have always communicated within corporations across and sometimes in flagrant disregard for formal organizational structures. In response to post-industrial challenges, business are trying to unleash innovation by encouraging rather than stifling complex-systems-style communication and collaboration. Management consultants are calling for the outright replacement of hierarchical corporate structures with ones that are flatter, more fluid, and more network-oriented.

The cloud is a prime example of complexity within the digital realm. A small business may run its finances using an online invoicing service from one company, an expense service from another, and a tax service from a third. Each of those companies may in turn leverage lower-level cloud services that are invisible to the end customer. If, for example, Amazon Web Services (AWS) has an outage, does the small business need to worry? They may not know that their invoicing service runs on top of Heroku's Platform-as-a-Service (PaaS). Even if they do, they still might not know that Heroku runs on top of AWS.

21st-century workplace trends are fundamentally changing the relationship between companies and employees. So-called Bring Your Own Device (BYOD) means that employees own their own laptops and smartphones, and can mix per-

sonal and company data on the same devices. Telecommuting and coworking move physical workspaces out of companies' control. Rogue IT lets employees buy and manage computing services without IT's control or even knowledge. Finally, companies' growing reliance on freelance labor footnote: [labor experts estimate that 40% of the U.S. workforce will consist of freelancers by the year 2020] changes the company-employee relationship at the most basic level.

Together, these trends all contribute to transforming corporate environments from complicated to complex systems. They transform the management of people, devices, systems, and data from a closed hierarchy to an open network. Open organizational networks create new opportunities for business resilience, adaptability, and creativity. At the same time, though, they stress traditional management practices based on control and stability.

From Efficiency to Adaptability

20th century business structures epitomized the model of corporations as complicated systems. Companies flourished by growing in both size and structure. They mastered industrial-era technical and management practices that maximized efficiency and stability. They used these practices to create robust hierarchies that supported ever-increasing economies of scale.

Giant, long-lived companies like Ford, Johnson and Johnson, AT&T, IBM, and Kodak created and dominated entire product categories.. They became known as blue chip stocks, with "a reputation for quality, reliability, and the ability to operate profitably in good times and bad" [4].

The 21st century is featuring the arrival of disruption. Companies like Salesforce, Apple, and Tesla compete, not by beating their rivals at their own game, but by changing the game itself. Software-as-a-Service (SaaS) doesn't just retool software for a service economy. It challenges the very relevancy of on-premise software. Why bear the cost or burden of installing and operating your own software when you can let a service provider do it for you? Why incur large up-front capital expenditures when you can pay on-demand service fees that ebb and flow with your business?

Smartphones with high-resolution, built-in cameras challenge the relevancy of the camera as a dedicated product. Why carry around a separate photo-taking device when you can use the one that's already in your pocket? Why fumble around with SD cards to transfer your photos to a computer in order to edit and share it, when

4. *http://en.wikipedia.org/wiki/Blue_chip_%28stock_market%29*

you can edit and upload directly from the device that took the picture in the first place? Kodak's inability to adapt to the smartphone revolution doomed it to a rapid death after a hundred years of operation.

Tesla doesn't just compete with companies that produce internal combustion-engine cars. Those companies are also producing cars with electric engines. By focusing on the zero-maintenance nature of electric cars, and selling them directly to customers, Tesla is challenging the very existence of physical sales-and-service dealerships. The car dealership industry is responding, not by innovating their own practices, but by lobbying state legislatures to forbid direct sales of automobiles from manufacturers to customers. This tactic represents the competitive strategy of the beleaguered.

Economic transitions are fecund opportunities for disruption. As we move from industrialism to post-industrialism, new companies are disrupting incumbents by better understanding and grabbing hold of the nature of service and digital infusion. Smartphones replace cameras, not just because of their physical convenience, but also because of their native integration with digital services in the cloud.

Cameras were invented during the era of physics and chemistry. They represented insights into the nature of light and glass and silver and paper. Digital photography dispenses with negatives and prints in favor of pixels. By doing so, it integrates photography with the realms of content and social media. It dissolves the boundary between pointing a physical device at an interesting building, and chatting with your friends back at home about your trip to Europe. Facebook believed in the power of infusing photography and social media enough to pay $1 billion for a mobile photo uploading application.

Toyota and Chevrolet approached electric cars as a matter of replacing drivetrains. Tesla, on the other hand, took the opportunity to reimagine the entire experience of owning and operating a cars. Tesla's challenge to the dealership model strikes at the heart of one of the most unsatisfying service experiences in modern life. Their cars are also deeply deeply digitally infused. Their onboard displays look more like an iPad than a traditional car dashboard. They offer an iPhone app that lets drivers remotely open vents and lock and unlock the car.

In a testament to the "software is eating the world" meme [5], Tesla engineers their cars to work like software as much as hardware. They provide API's that allow third parties (including technically-minded owners) to write their own applications. They remotely update their cars' onboard computer systems the way one would a

5. http://online.wsj.com/news/articles/SB10001424053111903480904576512250915629460

website. API's and automated updates may not sound strange or unusual in this day and age, until we remember it's a car we're talking about!

FACING DISRUPTION

The pace of disruption is accelerating. Kodak was in business for 100 years before being disrupted by digital cameras and smartphones. Microsoft went from a monopoly to an also-ran in the course of 20 years. In 2012 Nokia was the world's largest cell phone manufacturer. In 2013, when Microsoft bought them, pundits wondered whether one irrelevant company was buying another. Apple went from the world's most valuable company to a question mark - are they being disrupted by Google? Is Samsung beating them? Have they lost their mojo? - in 12 months.

Disruption invokes what Clay Christensen called the "innovator's dilemma". Disruptive innovations address customer needs left unserved by incumbents. These needs are unserved precisely because doing so would be unprofitable given the existing business model. Incumbents are often paralyzed against responding to disruptive competition because they can't get out from under their own feet.

In order to succeed in the age of disruption, companies must change their basic approach. They need to shift their emphasis from perpetuating stability to disrupting themselves. Instead of excelling at doing the same things better, faster, and cheaper, they need to challenge themselves to continually do different things, and continually do them differently. They need to learn to value learning over success, to value the ability to change direction over the ability to maintain course. In other words, they need to shift their core competency from efficiency to adaptability.

Apple is the poster child for the growing sensitivity to disruption in the marketplace. Investors no longer judge Apple primarily by its revenues, or profits, or growth, or market share. They judge the company by what it's done that's new and different. Apple successfully transformed itself from a computer company to a mobile device company. Now, though, announcing a new, better, faster, bigger iPhone isn't enough. Investors and pundits clamor for a TV, or a phone, or even better, something that doesn't have a name yet. The expectation isn't that Apple will improve the next device they release, but rather that they'll redefine it the way they did the cellphone.

Brands As Digital Conversations

Post-industrialism impacts companies on every level. They must truly become digital businesses. They must transform, not just how they operate or organize themselves, but also how they conceive of themselves. The post-industrial worldview

must inform everything they do, from their most basic daily processes, all the way up to the way they perceive themselves and behave as brands.

A brand represents "the unique story that consumers recall when they think of you" [6]. That story reflects the promises you've made to your customers, and the extent to which you've kept or and broken them. These promises involve commitments to help customers, and can operate on multiple levels. A sneaker company may promise to keep your feet comfortable while you run, but also to help you look cool. A car company may promise to help you drive safely, but also to maintain your social status.

Companies devote tremendous to creating and maintaining their brands. The post-industrial economy makes brand maintenance significantly more challenging. Service transforms it from a vendor-driven activity into a conversation with the customer. Service providers must make promises about listening and responding as much as making and delivering. I judge my car company as much by their service department as by the quality of the car itself. I also judge them by their ability to improve their products and services over time.

Digital infusion moves the brand conversation firmly into the digital realm. I've reached the point where 99% of my relationship with my bank happens via their website. My impression of my bank's brand derives directly from the quality of their online presence. If their website is slow, clumsy-looking, and hard to use, those characteristics will define the story I recall when I think of them. By making their site faster, better looking, and easier to use, they improve not just the quality of their online banking service, but also the quality of their brand.

Complexity challenges companies' ability to control their brand promises. When failure is inevitable, broken promises also become inevitable. Brand maintenance thus must incorporate the ability to repair promises in addition to keeping them in the first place. The way in which companies respond to events such as security breaches become critical brand quality moments.

Companies must do more than just keep their promises. They also must make the right promises. Promising speed and handling, when people value fuel economy as much as performance, may degrade rather than enhance a car company's brand. Disruption complicates brand maintenance by changing the landscape under companies' feet. Tesla, for example, is disrupting the meaning of brand in the automobile industry by combining performance, luxury, and environmental sensitivity.

6. Lean Branding

The post-industrial economy dissolves any remaining separation between branding and conducting business. Social media contributes to this trend by wresting control of a company's brand away from it in favor of consumers. No longer does the company drive the public's impression of it. Instead, they become mere participants in the discourse about their identity, quality, and value.

Post-industrialism turns brand management into a digital conversation between a company and its customers. Brand quality depends on a company's ability to conduct that conversation as seamlessly and empathically as possible. Having lost control of the message, companies must shift their focus from trying to shape their customers' perceptions, needs, and desires, to accurately understanding and responding to them. The capacity for empathic digital conversation thus becomes the defining characteristic of post-industrial business. The ability to power the digital brand conversation in turn becomes the defining measure of quality for post-industrial IT.

The New Business Imperative

In order to shift their approach to brand management from a broadcast model to a conversational model, 21st-century businesses must simultaneously transform the way they relate with their customers, and the way they organize their internal operations. Conversation depends on the ability to listen, and to respond appropriately to what you've heard. Digital businesses thus must organize themselves to accurately, continuously process market and customer feedback.

The ability to process feedback fluidly is a critical component of post-industrial business success. Service requires conversational marketing and co-creative business operations. Infusion requires deep integration between technical and business concerns, across physical and virtual dimensions. Complexity requires exploration and tolerance for failure. Disruption requires an unfettered ability to uncover and pursue new possibilities.

These requirements necessitate an internal transformation that mirrors the external one. In order to provide high-quality, digitally infused service, the entire delivery organization must function as an integrated whole. In order to handle the perturbations caused by complexity and disruption, that same organization must be able to flex and adapt. The post-industrial company thus begins to look more like an organism and less like a machine.

Post-industrial businesses are beginning to experiment with non-traditional organizational structures. Dave Gray has described networks of pods in his book, "The Connected Company". Zappo's, an online shoe retailer, is experimenting with

holacratic management structures that distribute decision making throughout self-organizing teams. Amazon CEO Jeff Bezos coined the phrase "two-pizza team" to refer to small, integrated teams that have the autonomy and intimacy necessary to move quickly. Yammer, an enterprise collaboration software vendor purchased by Microsoft in 2012, dynamically creates similarly sized functional teams. Each team dissolves and reorganizes after every project in order to propagate knowledge throughout the larger organization.

These new structures leverage the power of emergence to balance flexibility with coherency. In order to use them successfully, however, 21st-century businesses need more than a change in org chart. They need a new worldview from which to operate, one which shifts the emphasis of management values, goals, structures, and the IT systems that support them:

- From efficiency, scale, and stability to speed, flexibility, and nimbleness
- From discontinuous broadcast to continuous conversation
- From avoiding failure to absorbing it
- From success as accomplishment to success as learning

THE **LEAN** SERIES ERIC RIES, SERIES EDITOR

Jez Humble, Joanne Molesky & Barry O'Reilly

LEAN ENTERPRISE

How High Performance Organizations Innovate at Scale

O'REILLY®

CHAPTER 6

Deploy Continuous Improvement

> The paradox is that when managers focus on productivity, long-term improvements are rarely made. On the other hand, when managers focus on quality, productivity improves continuously.
>
> John Seddon

In most enterprises, there is a distinction between the people who build and run software systems (often referred to as "IT") and those who decide what the software should do and make the investment decisions (often called "the business"). These names are relics of a bygone age in which IT was considered a cost necessary to improve efficiencies of the business, not a creator of value for external customers by building products and services. These names and the functional separation have stuck in many organizations (as has the relationship between them, and the mindset that often goes with the relationship). Ultimately, we aim to remove this distinction. In high-performance organizations today, people who design, build, and run software-based products are an integral part of business; they are given—and accept—responsibility for customer outcomes. But getting to this state is hard, and it's all too easy to slip back into the old ways of doing things.

Achieving high performance in organizations that treat software as a strategic advantage relies on *alignment* between the IT function and the rest of the organization, along with the ability of IT to *execute*. It pays off. In a report for the *MIT Sloan Management Review*, "Avoiding the Alignment Trap in Information Technology," the authors surveyed 452 companies and discovered that

high performers (7% of the total) spent a little less than average on IT while achieving substantially higher rates of revenue growth.[1]

However, *how* you move from low performance to high performance matters. Companies with poor alignment and ineffective IT have a choice. Should they pursue alignment first, or try to improve their ability to execute? The data shows that companies whose IT capabilities were poor achieve worse results when they pursue alignment with business priorities before execution, even when they put significant additional investment into aligned work. In contrast, companies whose engineering teams do a good job of delivering their work on schedule and simplifying their systems achieve better business results with much lower cost bases, even if their IT investments aren't aligned with business priorities.

The researchers concluded that to achieve high performance, companies that rely on software should focus first and foremost on their ability to execute, build reliable systems, and work to continually reduce complexity. Only then will pursuing alignment with business priorities pay off.

However, in every team we are always balancing the work we do to improve our capability against delivery work that provides value to customers. In order to do this effectively, it's essential to manage both kinds of work at the program and value stream levels. In this chapter we describe how to achieve this by putting in place a framework called *Improvement Kata*. This is the first step we must take to drive continuous improvement in our execution of large scale programs. Once we have achieved this, we can use the tools in the following chapters to identify and remove no-value-add activity in our product development process.

The HP LaserJet Firmware Case Study

We will begin with a case study from the HP LaserJet Firmware team, which faced a problem with both alignment and execution.[2] As the name suggests, this was a team working on embedded software, whose customers have no desire to receive software updates frequently. However, it provides an excellent example of how the principles described in the rest of Part III work at scale in a distributed team, as well as of the economic benefits of adopting them.

HP's LaserJet Firmware division builds the firmware that runs all their scanners, printers, and multifunction devices. The team consists of 400 people distributed across the USA, Brazil, and India. In 2008, the division had a

1 [schpilberg]

2 This case study is taken from [gruver], supplemented by numerous discussions with Gary Gruver.

problem: they were moving too slowly. They had been on the critical path for all new product releases for years, and were unable to deliver new features: "Marketing would come to us with a million ideas that would dazzle the customer, and we'd just tell them, 'Out of your list, pick the two things you'd like to get in the next 6–12 months.'" They had tried spending, hiring, and outsourcing their way out of the problem, but nothing had worked. They needed a fresh approach.

Their first step was to understand their problem in greater depth. They approached this by using *activity accounting*—allocating costs to the activities the team is performing. Table 6-1 shows what they discovered.

Table 6-1. Activities of the HP LaserJet Firmware team in 2008

% of costs	Activity
10%	Code integration
20%	Detailed planning
25%	Porting code between version control branches
25%	Product support
15%	Manual testing
~5%	Innovation

This revealed a great deal of no-value-add activity in their work, such as porting code between branches and detailed upfront planning. The large amount spent on current product support also indicated a problem with the quality of the software being produced. Money spent on support is generally serving *failure demand*, as distinct from *value demand*, which was only driving 5% of the team's costs.[3]

The team had a goal of increasing the proportion of spending on innovation by a factor of 10. In order to achieve that goal, they took the bold but risky decision to build a new firmware platform from scratch. There were two main architectural goals for the new "FutureSmart" platform. The first goal was to increase quality while reducing the amount of manual testing required for new

3 The distinction between failure demand and value demand comes from John Seddon, who noticed that when banks outsourced their customer service to call centers, the volume of calls rose enormously. He showed that up to 80% of the calls were "failure demand" of people calling back because their problems were not solved correctly the first time [seddon].

firmware releases (a full manual testing cycle required six weeks). The team hoped that this goal could be achieved through:

- The practice of continuous integration (which we describe in Chapter 8)
- Significant investment in test automation
- Creating a hardware simulator so that tests could be run on a virtual platform
- Reproduction of test failures on developer workstations

Three years into the development of the new firmware, thousands of automated tests had been created.

Second, they wanted to remove the need for the team to spend time porting code between branches (25% of total costs on the existing system). This was caused by the need to create a branch—effectively a copy of the entire codebase—for every new line of devices under development. If a feature or bug-fix added to one line of devices was required for any others, these changes would need to be merged (copied back) into the relevant code branches for the target 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 run on any device, removing the need to use version control branches to handle the differences between devices.

The final goal of the team was to reduce the amount of time its members spent on detailed planning activities. The divisions responsible for marketing the various product lines had insisted on detailed planning because they simply could not trust the firmware team to deliver. Much of this time was spent performing detailed re-plans after failing to meet the original plans.

Furthermore, the team did not know how to implement the new architecture, and had not used trunk-based development or continuous integration at scale before. They also understood that test automation would require a great deal of investment. How would they move forward?

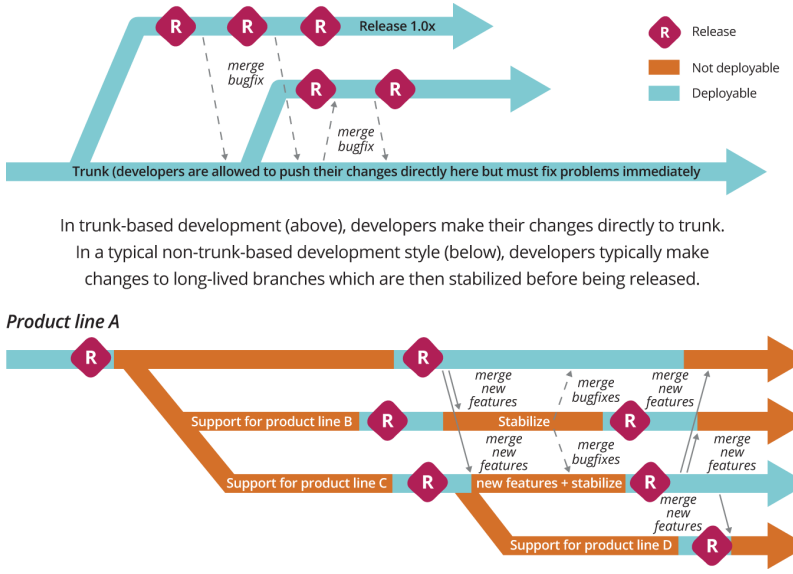


Figure 6-1. *Branching versus trunk-based development*

It's all too easy to turn a sequence of events into a story in an attempt to explain the outcome—a cognitive bias that Nassim Taleb terms the *narrative fallacy*. This is, arguably, how methodologies are born. What struck us when studying the FutureSmart case were the similarities between the program management method of FutureSmart's engineering management team and the approach Toyota uses to manage innovation as described in Mike Rother's *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results*.[4]

Drive Down Costs Through Continuous Process Innovation Using the Improvement Kata

The Improvement Kata, as described by Mike Rother, is a general-purpose framework and a set of practice routines for reaching goals where the path to the goal is uncertain. It requires us to proceed by iterative, incremental steps, using very rapid cycles of experimentation. Following the Improvement Kata also increases the capabilities and skills of the people doing the work, because it requires them to solve their own problems through a process of continuous experimentation, thus forming an integral part of any learning organization.

4 [rother-2010]

Finally, it drives down costs through identifying and eliminating waste in our processes.

The Improvement Kata needs to be first adopted by the organization's management, because it is a management philosophy that focuses on developing the capabilities of those they manage, as well as on enabling the organization to move towards its goals under conditions of uncertainty. Eventually, everybody in the organization should be practicing the Improvement Kata habitually to achieve goals and meet challenges. This is what creates a culture of continuous improvement, experimentation, and innovation.

To understand how this works, let's examine the concept of *kata* first. A kata is "a routine you practice deliberately, so its pattern becomes a habit."[5] Think of practicing scales to develop muscle memory and digital dexterity when learning the piano, or practicing the basic patterns of movement when learning a martial art (from which the term derives), or a sport. We want to make continuous improvement a habit, so that when faced with an environment in which the path to our goal is uncertain, we have an instinctive, unconscious routine to guide our behavior.

In Toyota, one of the main tasks of managers is to teach the Improvement Kata pattern to their teams and to facilitate running it (including coaching learners) as part of everyday work. This equips teams with a method to solve their own problems. The beauty of this approach is that if the goal or our organization's environment changes, we don't need to change the way we work—if everybody is practicing the Improvement Kata, the organization will automatically adapt to the new conditions.

The Improvement Kata has four stages that we repeat in a cycle, as shown in Figure 6-2.

5 [rother]

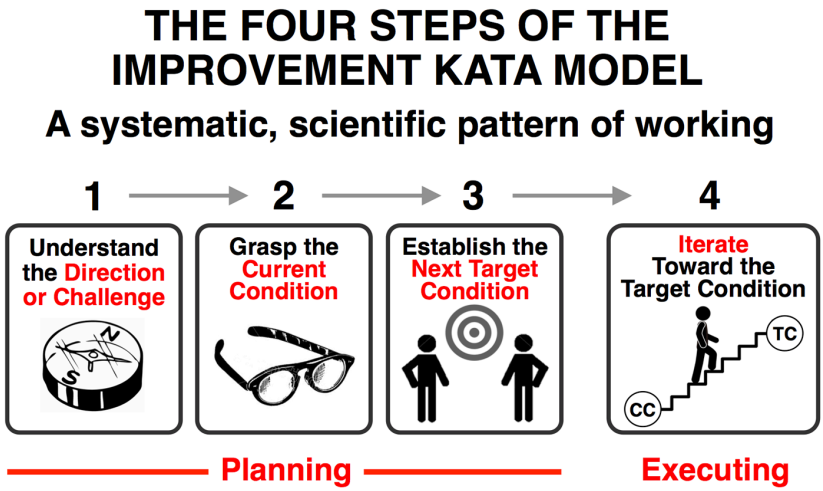


Figure 6-2. *The Improvement Kata, courtesy of Mike Rother*

Understand the Direction

We begin with understanding the direction. Direction is derived from the vision set by the organization's leadership. A good vision is one that is inspiring—and, potentially, unattainable in practice. For example, the long-term vision for Toyota's production operations is "One-piece flow at lowest possible cost." In *Leading Lean Software Development*, Mary and Tom Poppendieck describe Paul O'Neill setting the objective for Alcoa to be "Perfect safety for all people who have anything to do with Alcoa" when he became CEO in 1987.[6]

People need to understand that they must always be working towards the vision and that they will never be done with improvement. We will encounter problems as we move towards the vision. The trick is to treat them as obstacles to be removed through experimentation, rather than objections to experimentation and change.

Based on our vision and following the Principle of Mission, we must understand the direction we are working in, at the level of the whole organization and at the value stream level. This challenge could be represented in the form of a future-state value stream map (see Chapter 7 for more on value stream mapping). It should result in a measurable outcome for our customers, and we should plan to achieve it in six months to three years.

6 [poppendieck-09], Frame 13, "Visualize Perfection."

Planning: Grasp the Current Condition and Establish a Target Condition

After we have understood the direction at the organizational and value stream levels, we incrementally and iteratively move towards it at the process level. Rother recommends setting target conditions with a horizon between one week and three months out, with a preference for shorter horizons for beginners. For teams that are using iterative, incremental methods to perform product development, it makes sense to use the same iteration (or sprint) boundaries for both product development and Improvement Kata iterations. Teams that use flow-based methods such as the Kanban Method (for which see Chapter 7) and continuous delivery (described in Chapter 8) can create Improvement Kata iterations at the program level.

As with all iterative product development methods, Improvement Kata iterations involve a planning part and an execution part. Here, planning involves grasping the current condition at the process level and setting a target condition that we aim to achieve by the end of the next iteration.

Analyzing the current condition "is done to obtain the facts and data you need in order to then describe an appropriate next target condition. What you're doing is trying to find the current pattern of operation, so you can establish a desired pattern of operation (a target condition)." The target condition "describes in measurable detail how you want a process to function…[It is] a description and specification of the operating pattern you want a process or system to have on a future date."[7]

The team grasps the current condition and establishes a target condition together. However, in the planning phase the team does not plan how to *move* 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.

A target condition identifies the process being addressed, sets the date by which we aim to achieve the specified condition, and specifies measurable details of the process as we want it to exist. Examples of target conditions include WIP (work in progress) limits, the implementation of Kanban or a continuous integration process, the number of good builds we expect to get per day, and so forth.

7 [rother]

Getting to the Target Condition

Since we are engaging in process innovation in conditions of uncertainty, we cannot know in advance how we will achieve the target condition. It's up to the people doing the work to run a series of experiments using the Deming cycle (plan, do, check, act), as described in Chapter 3. The main mistakes people make when following the Deming cycle are performing it too infrequently and taking too long to complete a cycle. With Improvement Kata, everybody should be running experiments on a daily basis.

Each day, people in the team go through answering the following five questions:[8]

1. What is the target condition?
2. What is the actual condition now?
3. What obstacles do you think are preventing you from reaching the target condition? Which one are you addressing now?
4. What is your next step? (Start of PDCA cycle.) What do you expect?
5. When can we go and see what we learned from taking that step?

As we continuously repeat the cycle, we reflect on the last step taken to introduce improvement. What did we expect? What actually happened? What did we learn? We might work on the same obstacle for several days.

This experimental approach is already central to how engineers and designers work. Designers who create and test prototypes to reduce the time taken by a user to complete a task are engaged in exactly this process. For software developers using test-driven development, every line of production code they write is essentially part of an experiment to try and make a unit test pass. This, in turn, is a step on the way to improving the value provided by a program—which can be specified in the form of a target condition, as we describe in Chapter 9.

The Improvement Kata is simply a generalization of this approach to improvement, combined with applying it at multiple levels of the organization, as we discuss when presenting strategy deployment in Chapter 15.

How the Improvement Kata Differs from Other Methodologies

You can think of the Improvement Kata as a *meta-methodology* since it does not apply to any particular domain, nor does it tell you what to do. It is not a playbook; rather, as with the Kanban Method, it teaches teams how to *evolve*

8 [rother]

their existing playbook. In this sense, it differs from other agile frameworks and methodologies. With the Improvement Kata, there is no need to make existing processes conform to those specified in the framework; process and practices you use are *expected* to evolve over time. *This* is the essence of agile: teams do not become agile by adopting a methodology. Rather, true agility means that teams are constantly working to evolve their processes to deal with the particular obstacles they are facing at any given time.

NOTE

Single-Loop Learning and Double-Loop Learning

Changing the way we think and behave in reaction to a failure is crucial to effective learning. This is what distinguishes *single-loop learning* from *double-loop learning* (see Figure 6-3). These terms were coined by management theorist Chris Argyris, who summarizes them as follows: "When the error detected and corrected permits the organization to carry on its present policies or achieve its present objectives, then that error-and-correction process is single-loop learning. Single-loop learning is like a thermostat that learns when it is too hot or too cold and turns the heat on or off. The thermostat can perform this task because it can receive information (the temperature of the room) and take corrective action. Double-loop learning occurs when error is detected and corrected in ways that involve the modification of an organization's underlying norms, policies and objectives."[9] Argyris argues that the main barrier to double-loop learning is defensiveness when confronted with evidence that we need to change our thinking, which can operate at both individual and organizational levels. We discuss how to overcome this anxiety and defensiveness in Chapter 11.

9 [argyris], pp. 2–3.

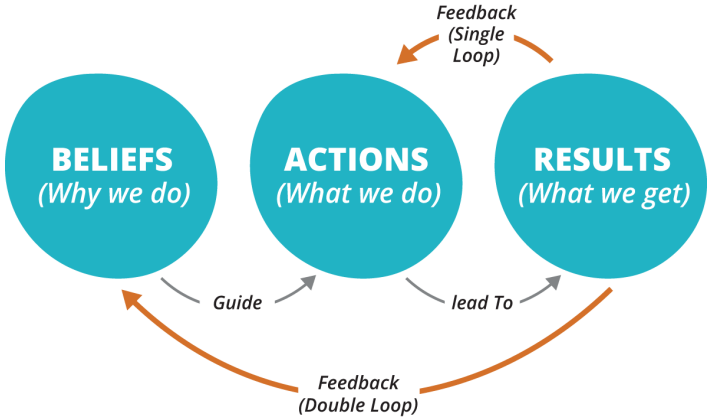


Figure 6-3. *Single-loop and double-loop learning*

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.

Deploying the Improvement Kata

Rother's work on the Improvement Kata was a direct result of his enquiry into how people become managers at Toyota. There is no formal training program, nor is there any explicit instruction. However, to become a manager at Toyota, one must have first worked on the shop floor and therefore participated in the Improvement Kata. Through this process, managers receive implicit training in how to manage at Toyota.

This presents a problem for people who want to learn to manage in this way or adopt the Improvement Kata pattern. It is also a problem for Toyota—which is aiming to scale faster than is possible through what is effectively an apprenticeship model for managers.

Consequently, Rother presents the Coaching Kata in addition to the Improvement Kata. It is part of deploying the Improvement Kata, but it is also as a way to grow people capable of working with the Improvement Kata, including managers.

Rother has made a guide to deploying the Improvement Kata, *The Improvement Kata Handbook*, available for free on his website at http://bit.ly/11iBzlY.

How the HP LaserJet Team Implemented the Improvement Kata

The direction set by the HP LaserJet leadership was to improve developer productivity by a factor of 10, so as to get firmware off the critical path for product development and reduce costs.[10] They had three high-level goals:

1. Create a single platform to support all devices
2. Increase quality and reduce the amount of stabilization required prior to release
3. Reduce the amount of time spent on planning

They did not know the details of the path to these goals and didn't try to define it. The key decision was to work in iterations, and set target conditions for the end of each four-week iteration. The target conditions for Iteration 30 (about 2.5 years into the development of the FutureSmart platform) are shown in Figure 6-4.

The first thing to observe is that the target conditions (or "exit criteria" as they are known in FutureSmart) are all measurable conditions. Indeed, they fulfill all the elements of SMART objectives: they are specific, measurable, achievable, relevant, and time bound (the latter by virtue of the iterative process). Furthermore, many of the target conditions were not focused on features to be delivered but on attributes of the system, such as quality, and on activities designed to validate these attributes, such as automated tests. Finally, the objectives for the entire 400-person distributed program for a single month was captured in a concise form that fit on a single piece of paper—similar to the standard A3 method used in the Toyota Production System.

How are the target conditions chosen? They are "aggressive goals the team feels are possible and important to achieve in 4 weeks…We typically drive hard for these stretch goals but usually end up hitting around 80% of what we thought we could at the beginning of the month."[11] Often, target conditions would be changed or even dropped if the team found that the attempt to achieve them results in unintended consequences: "It's surprising what you learn in a month and have to adjust based on discovery in development."[12]

10 [gruver], p. 144.

11 [gruver], p. 40.

12 Ibid.

Rank	Theme	Exit Criteria Objective met / *Objective not met*
0	Quality threshold	P1 issues open < 1 week L2 test failure 24 hour response
1	Quarterly bit release	*A) Final P1 change requests fixed* B) Reliability error rate at release criteria
2	New platform stability and test coverage	A) Customer Acceptance Test 100% passing B) All L2 test pillars 98% passing C) L4 test pillars in place D) L4 test coverage for all Product Turn On requirements E) 100% execution of L4 tests on new products
3	Product Turn On dependencies and key features	A) Print for an hour at speed to finisher with stapling B) Copy for an hour *at speed* *C) Enable powersave mode* D) Manufacturing nightly test suite execution E) Common Test Library support for four-line control panel display
4	Build for next-gen products	*A) End-to-end system build on new processor* *B) High-level performance analysis on new processor*
5	Fleet integration plan	Align on content and schedule for "slivers" of end-to-end agile test with system test lab

Figure 6-4. *Target conditions for iteration 30*[13]

13 Gruver, Gary, Young, Mike, Fulghum, Pat. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*, 1st Edition, (c) 2013. Reprinted by permission of Pearson Education, Inc. Upper Saddle River, NJ.

WARNING

What Happens When We Do Not Achieve Our Target Conditions?

In bureaucratic or pathological organizational cultures, not achieving 100% of the specified target conditions is typically considered a failure. In a generative culture, however, we *expect* to not be able to achieve all our target conditions. The purpose of setting aggressive target conditions is to reveal obstacles so we can overcome them through further improvement work. Every iteration should end with a retrospective (described in Chapter 11) in which we investigate how we can get better. The results form part of the input for the next iteration's target conditions. For example, if we fail to achieve a target condition for the number of good builds of the system per day, we may find that the problem is that it takes too long to provision test environments. We may then set a target condition to reduce this in the next iteration.

This approach is a common thread running through all of Lean Thinking. The subtitle of Mary and Tom Poppendieck's book *Leading Lean Software Development* reads: "Results are not the point." This is a provocative statement that gets to the heart of the lean mindset. If we achieve the results by ignoring the process, we do not learn how to improve the process. If we do not improve the process, we cannot repeatably achieve better results. Organizations that put in place unmodifiable processes that everybody is required to follow, but which get bypassed in a crisis situation, fail on both counts.

This adaptive, iterative approach is not new. Indeed it has a great deal in common with what Tom Gilb proposed in his 1988 work *Principles of Software Engineering Management*:[14]

> We must set measurable objectives for each next small delivery step. Even these are subject to constant modification as we learn about reality. It is simply not possible to set an ambitious set of multiple quality, resource, and functional objectives, and be sure of meeting them all as planned. We must be prepared for compromise and trade-off. We must then design (engineer) the immediate technical solution, build it, test it, deliver it—and get feedback. This feedback must be used to modify the immediate design (if necessary), modify the major architectural ideas (if necessary), and modify both the short-term and the long-term objectives (if necessary).

14 [gilb-88], p. 91.

Designing for Iterative Development

In large programs, demonstrating improvement within an iteration requires ingenuity and discipline. It's common to feel we can't possibly show significant progress within 2–4 weeks. Always try to find something small to bite off to achieve a little bit of improvement, instead of trying to do something you think will have more impact but will take longer.

This is not a new idea, of course. Great teams have been working this way for decades. One high-profile example is the Apple Macintosh project where a team of about 100 people—co-located in a single building—designed the hardware, operating system, and applications for what was to become Apple's breakthrough product.

The teams would frequently integrate hardware, operating system, and software to show progress. The hardware designer, Burrell Smith, employed programmable logic chips (PALs) so he could prototype different approaches to hardware design rapidly in the process of developing the system, delaying the point at which it became fixed—a great example of the use of optionality to delay making final decisions.[15]

After two years of development, the new firmware platform, FutureSmart, was launched. As a result, HP had *evolved* a set of processes and tools that substantially reduced the cost of no-value-add activities in the delivery process while significantly increasing productivity. The team was able to achieve "predictable, on-time, regular releases so new products could be launched on time."[16] Firmware moved off the critical path for new product releases for the first time in twenty years. This, in turn, enabled them to build up trust with the product marketing department.

As a result of the new relationship between product marketing and the firmware division, the FutureSmart team was able to considerably reduce the time spent on planning. Instead of "committing to a final feature list 12 months in advance that we could never deliver due to all the plan changes over the time,"[17] they looked at each planned initiative once every 6 months and did a 10-minute estimate of the number of months of engineering effort required for a given initiative, broken down by team. More detailed analysis would be performed once work was scheduled into an iteration or mini-milestone. An example of the output from one of these exercises is shown in Figure 6-5.

15 *http://folklore.org/StoryView.py?story=Macintosh_Prototypes.txt*

16 [gruver], p. 89.

17 [gruver], p. 67.

		High-Level Estimate – FW Engineering Months											
Rank	*Initiative*	*Component 1* (25-30)	*Component 2* (20-25)	*Component 3* (30-40)	*Component 4* (30-40)	*Component 5* (20-30)	*Component 6* (20-30)	*Component 7* (20-30)	*Component 8* (15-25)	*Component 10* (40-50)	*Component 11* (20-30)	*Component 12* (20-30)	*other teams*	**TOTAL**
1	Initiative A			*21*			*5*	*3*		*1*				***30***
2	Initiative B	*3*							*4*				*17*	***24***
3	Initiative C		*5*							*2*	*1*	*1*		***9***
4	Initiative D							*10*		*2*	*2*	*2*		***16***
5	Initiative E					*20*						*3*	*5*	***28***
6	Initiative F	*23*							*5*	*6*			*2*	***36***
7	Initiative G									*2*				***2***
8	Initiative H											*5*		***5***
9	Initiative I												*3*	***3***
10	Initiative J		*20*	*27*			*17*			*39*	*17*	*21*	*9*	***150***
11	Initiative K			*3*	*30*		*3*		*3*	*14*			*12*	***65***
12	Initiative L									*2*				***2***
13	Initiative M	*3*						*10*		*6*	*6*	*6*		***31***
		29	*25*	*51*	*30*	*20*	*25*	*23*	*12*	*74*	*26*	*38*	*59*	***401***

Figure 6-5. *Ballpark estimation of upcoming initiatives*[18]

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.

Ultimately the most important test of the planning process is whether we are able to keep the commitments we make to our stakeholders, including end users. As we saw, a more lightweight planning process resulted in firmware development moving off the critical path, while at the same time reducing both development costs and failure demand. Since we would expect failure demand to *increase* as we increase throughput, this is doubly impressive.

18 Gruver, Gary, Young, Mike, Fulghum, Pat. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*, 1st Edition, (c) 2013. Reprinted by permission of Pearson Education, Inc. Upper Saddle River, NJ.

Three years after their initial measurements, a second activity-accounting exercise offered a snapshot of the results the FutureSmart team had achieved with their approach, shown in Table 6-2.

Table 6-2. Activity of the HP LaserJet Firmware Team in 2011

% of costs	Activity	Previously
2%	Continuous integration	10%
5%	Agile planning	20%
15%	One main branch	25%
10%	Product support	25%
5%	Manual testing	15%
23%	Creating and maintaining automated test suites	0%
~40%	Innovation	~5%

Overall, the HP LaserJet Firmware division changed the economics of the software delivery process by adopting continuous delivery, comprehensive test automation, an iterative and adaptive approach to program management, and a more agile planning process.

Economic Benefits of HP FutureSmart's Agile Transformation

- Overall development costs were reduced by ~40%.
- Programs under development increased by ~140%.
- Development costs per program went down 78%.
- Resources driving innovation increased eightfold.

The most important point to remember from this case study is that the enormous cost savings and improvements in productivity were only possible on the basis of a large and ongoing *investment* made by the team in test automation and continuous integration. Even today, many people think that Lean is a management-led activity and that it's about simply *cutting costs*. In reality, it requires *investing* to remove waste and reduce failure demand—it is a worker-led activity that, ultimately, can continuously drive down costs and improve quality and productivity.

Managing Demand

Up to now, we've been discussing how to improve the throughput and quality of the delivery process. However, it is very common for this kind of improvement work to get crowded out by business demands, such as developing new features. This is ironic, given that the whole purpose of improvement work is to increase the rate at which we can deliver as well as the quality of what gets delivered. It's often hard to make the outcome of improvement work tangible—which is why it's important to make it visible by activity accounting, including measuring the cycle time and the time spent serving failure demand such as rework.

The solution is to use the same mechanism to manage both demand and improvement work. One of the benefits of using the Improvement Kata approach is that it creates alignment to the outcomes we wish to achieve over the next iteration across the whole program. In the original Improvement Kata, the target conditions are concerned with process improvement, but we can use them to manage demand as well.

There are two ways to do this. In organizations with a generative culture (see Chapter 1), we can simply specify the desired business goals as target conditions, let the teams come up with ideas for features, and run experiments to measure whether they will have the desired impact. We describe how to use impact mapping and hypothesis-driven development to achieve this in Chapter 9. However, more traditional enterprises will typically have a backlog of work prioritized at the program level by its lines of business or by product owners.

We can take a few different approaches to integrating a program-level backlog with the Improvement Kata. One possibility is for teams working within the program to deploy the Kanban Method, as described in Chapter 7. This includes the specification of work in process (WIP) limits which are owned and managed by these teams. New work will only be accepted when existing work is completed (where "completed" means it is at least integrated, fully tested with all test automation completed, and shown to be deployable).

TIP

Managing Cross-Cutting Work

Implementing some features within a program will involve multiple teams working together. To achieve this, the HP FutureSmart division would set up a small, temporary "virtual" feature team whose job is to coordinate work across the relevant teams.

The HP FutureSmart program, some of whose teams were using Scrum, took the approach of specifying a target velocity at the *program* level. Work adding up to the target velocity was accepted for each iteration, approximating a WIP limit. In order to implement this approach, all work was analyzed and estimated at a high level before being accepted. Analysis and estimation was kept to the bare minimum required to be able to consistently meet the overall program-level target conditions, as shown in Figure 6-5.

WARNING

Do Not Use Team Velocity Outside Teams

It is important to note that specifying a target velocity at the program level does *not* require that we attempt to measure or manage velocity at the team level, or that teams must use Scrum. Program-level velocity specifies the expected work capacity of all teams based on high-level estimates, as shown in Figure 6-5. If a team using Scrum accepts work based on these high-level feature specifications, they then create lower-level stories with which to work.

Scrum's team-level velocity measure is not all that meaningful outside of the context of a particular team. Managers should *never* attempt to compare velocities of different teams or aggregate estimates across teams. Unfortunately, we have seen team velocity used as a measure to compare productivity between teams, a task for which it is neither designed nor suited. Such an approach may lead teams to "game" the metric, and even to stop collaborating effectively with each other. In any case, it doesn't matter how many stories we complete if we don't achieve the business outcomes we set out to achieve in the form of program-level target conditions.

In this and the next chapter, we describe a much more effective way to measure progress and manage productivity—one that does not require all teams to use Scrum or "standardize" estimates or velocity. We use activity accounting and value stream mapping (described in Chapter 7) to measure productivity, and we use value stream mapping combined with the Improvement Kata to increase it—crucially, at the value stream level rather than at the level of individual teams. We measure and manage progress through the use of target conditions at the program level, and if we need to increase visibility, we reduce the duration of iterations.

Creating an Agile Enterprise

Many organizations look to try and adopt agile methods to improve the productivity of their teams. However, agile methods were originally designed around small, cross-functional teams, and many organizations have struggled to use these methods at scale. Some frameworks for scaling agile focus on creating such small teams and then adding structures to coordinate their work at the program and portfolio level.

Gary Gruver, Director of Engineering for FutureSmart, contrasts this approach of "trying to enable the efficiencies of small agile teams in an enterprise" with the FutureSmart team's approach of "trying to make an enterprise agile using the basic agile principles."[19] In the FutureSmart approach, while the teams ran within tight guide rails in terms of engineering practices (which we discuss in more detail in Chapter 8), there was relatively little attention paid to whether they had, for example, implemented Scrum at the team level. Instead, teams have relative autonomy to choose and evolve their own processes, provided they are able to meet the program-level target conditions for each iteration.

This required that engineering management had the freedom to set their own program-level objectives. That is, they didn't have to get budget approval to pay for process improvement work such as test automation or building out the toolchain for continuous integration. Indeed, the business wasn't even consulted on this work. All business demand was also managed at the program level. Notably, product marketing requests always went through the program-level process, without feeding work directly to teams.

Another important consideration is the way enterprises treat metrics. In a control culture, metrics and targets are often set centrally and never updated in response to the changes in behavior they produce. Generative organizations don't manage by metrics and targets. Instead, the FutureSmart management "use[s] the metrics to understand where to have conversations about what is not getting done."[20] This is part of the strategy of "Management by Wandering Around" pioneered by HP founders Bill Hewlett and Dave Packard.[21] Once we discover a problem, we ask the team or person having a hard time what we can do to help. We have discovered an opportunity to improve. If people are punished for failing to meet targets or metrics, one of the fallouts is that they 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.

Conclusion

The Improvement Kata provides a way to align teams and, more generally, organizations by taking goals and breaking them down into small, incremental outcomes (target conditions) that get us closer to our goal. The Improvement

19 [gruver], Chapter 15.

20 [gruver], p. 38.

21 Perhaps it's better characterized as "Management by Wandering Around and Asking Questions." In the Toyota Production System, this is known as a *gemba* walk.

Kata is not just a meta-methodology for continuous improvement at the enterprise and program level; it is a way to push ownership for achieving those outcomes to the edges of the organization, following the Principle of Mission. As we show in Chapter 9, it can also be used to run large programs of work.

The key characteristics of the Improvement Kata are its iterativeness and the ability to drive an experimental approach to achieve the desired target conditions, which makes it suitable for working in conditions of uncertainty. The Improvement Kata is also an effective way to develop the capabilities of people throughout the enterprise so they can self-organize in response to changing conditions.

The FutureSmart case study shows how a large, distributed team applied the Improvement Kata meta-method to increase productivity eightfold, improving quality and substantially reducing costs. The processes and tools the team used to achieve this transformation changed and evolved substantially over the course of the project. This is characteristic of a truly agile organization.

Implementing an enterprise-level continuous improvement process is a prerequisite for any ongoing large-scale transformation effort (such as adopting an agile approach to software delivery) at scale. True continuous improvement never ends because, as our organization and environment evolve, we find that what works for us today will not be effective when conditions change. High-performance organizations are constantly evolving to adapt to their environment, and they do so in an organic way, not through command and control.

Questions for readers:

- Do you know how much time your engineering organization is spending on no-value-add activities and servicing failure demand versus serving value demand, and what the major sources of waste are?
- Must engineering teams get permission to invest in work that reduces waste and no-value-add activity across the value stream as a whole, such as build, test, and deployment automation and refactoring? Are such requests denied for reasons such as “there is no budget” or “we don’t have time”?
- Does everyone within the organization know the short- and long-term outcomes they are trying to achieve? Who decides these outcomes? How are they set, communicated, reviewed, and updated?
- Do teams in your organization regularly reflect on the processes they use and find ways to experiment to improve them? What feedback loops are in place to find out which ideas worked and which didn’t? How long does it take to get this feedback?

O'REILLY®

DevOps in Practice

J. Paul Reed

Introduction

Practice makes perfect.

It's an adage we hear from an early age, usually around the time we start learning to tie our shoes, ride a bike, or play an instrument. As DevOps gets ready to celebrate its fifth birthday,[1] DevOps practitioners and the movement itself are starting to hear this familiar phrase.

It can be easy to forget that deliberately practicing a skill to hone and make our own is a time-honored technique. It can be hard to find the time for the necessary focused practice, as work, family, projects, and circumstance all impact our ability to find the time and space to do so. It can also be difficult when that "we" is a large organization, comprised of many different facets and personalities, with various motivations and incentives floating about.

Contained herein are two stories of organizations figuring out what "DevOps" means to them. Based on a series of interviews with people at different levels of the organization and working on various teams, we get to see them undertake the tasks of discovering what DevOps means in the context of their own organizational cultures. We also get to see them wrestle with how it looks functionally within their companies, expressed in the structure of their teams, and the path code takes from commit to customer. The characters in our story may surprise you, as they're not in the list of companies that generally come to mind when the phrase "DevOps posterchildren" is uttered.

1. Patrick Debois, widely considered to be the father of the word "DevOps," held the first DevOps Days in Ghent, Belgium, in October 2009.

Much is made of the fact that DevOps is about both "tools and culture! Tools *and* culture!" But as we shall see, while tools and culture are both important, perhaps the most important aspect to take note of is the journey itself.

What Is DevOps?

New to DevOps? Welcome! This book delves into the details of how two different organizations are working to become more "DevOps-like"; if you're unfamiliar with DevOps or would like to read more, we recommend:

- "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr", Velocity 2009 presentation by John Allspaw and Paul Hammond
- The Phoenix Project, by Gene Kim, Kevin Behr, and George Spafford
- *Building a DevOps Culture* (O'Reilly), free ebook by Mandi Walls

The organizations profiled also employ infrastructure as code and continuous delivery to accomplish their goals; these pieces give a more in-depth treatment of the fundamentals of those topics:

- Adam Jacob's chapter on "Infrastructure as Code" from *Web Operations* (O'Reilly), by John Allspaw and Jesse Robbins
- *Test-Driven Infrastructure with Chef, 2nd Edition*, by Stephen Nelson-Smith
- *Continuous Delivery*, by Jez Humble
- *Lean Enterprise*, by Jez Humble, Barry O'Reilly, and Joanne Molesky

Nordstrom

A Campout at the Colo

Rob Cummings hated deployments.

The year was 2004. Cummings was an operations engineer working on the team that supported *nordstrom.com*. After a bit of prodding, Cummings chuckles and admits that it is a bit odd. After all, it's not like it snuck up on him; getting code pushed out to production was a big part of any operations engineer's job in 2004.

By that point, large-scale retail websites were no longer a shiny new concept. The brick and mortars had started developing their online identities during the first dot-com boom of the early 2000s, as it became clear to a number of industries—sometimes viscerally—that this "World Wide Web thing" was not a fad and was very much not going away.

"It was a traditional environment," Cummings recalls: separate development and operations teams, operations peppered with myriad "throw-away shell scripts," anywhere from a few days to several weeks to provision compute resources for development and testing. The web applications were monoliths, deployed with a heavyweight process to facilitate the required heavy lifting, all frosted over with the amount of pageantry such a system implies. For its time, the team was doing a good job of meeting the business' needs; remember that new major versions of the browsers customers used to get to *nordstrom.com* were only released once a year back then. For the Nordstrom website, the company performed its major deployments about once a quarter or so, in a process they called "site-downs."

"I hated, *hated* site-downs, so much so I think I blocked most of them out of my mind," Cummings muses. Out of an abundance of caution, each site-down involved Nordstrom's operations engineers driving to the colocation facility to perform the deployment. Cummings recalls the amount of manual work to complete the deployment: "We'd just sit there all night and work on it until…it worked." If his description conjures up images of 2 am hacking sessions in the NOC fueled by pizza and Mountain Dew that so many of us lived through back then, think again: "We didn't even have that! The colo was out in the middle of nowhere, and it was the middle of the night. If you didn't bring any food, you weren't getting any food."

"It rarely went well," Cummings says. "But that was part of being in operations back then: you'd just figure it out."

Life in website development wasn't particularly easier, recalls Nordstrom's Courtney Kissler, who worked on the Website Engineering team. Even though a few years had passed, it was still the era of site-downs and heavyweight deployments; developers were trying to keep pace with the increased rate of change experienced in the latter half of the 2000s, working on features ever more furiously and trying to get them integrated and shipped ever more rapidly. "We had all these opposing forces; we had this long-standing throw-it-over-the-wall mentality, where the way we did things was just to give it to Rob's team and they'd figure it out." Kissler remembers a number of occasions where the operations team had to take point for figuring out why a feature was underperforming (or in worse cases, impacting a more noticeable part of the website). "Teams were staying up all night to get things going, and it was a pretty big morale hit." That made the job of developing features tough enough, but the real issue, Kissler said, which wasn't even clear to the team at the time: "Frankly, we were causing production outages and service interruptions" due to moving so quickly, yet so haphazardly. This translated into inconsistent releases, where about 30% of the time, features had to be turned dark after they'd gone live.

It was a tough period for those supporting the online customer experience, no matter which side of the site you were on.

The Event™

Organizations of a certain size and with a certain amount of history embed events within their consciousness. As stories of The Event™ are

passed down from manager to individual contributor and spread among the new hires, team by team, they become part of the company lore. They're given proper names. They serve as warnings to others, that "Here, there be dragons" and one team, many moons ago, wrestled that dragon. The goal is to help spread knowledge of what made the organization succeed (or what deficiency made it a bitter loss). The most pervasive Events become part of the institutional consciousness precisely because they contextualize the journey the organization and its people are themselves on today; it's the thing that spurred them to get on the road in the first place.

For Cummings, this event was the website falling over on one of the company's highest-volume days. "It was traditionally a very festive day." For both website developers and operators, this was the day to let their work from the past months shine, Cummings said: "The feeling was always 'Oh, we will watch all of this traffic coming to the site, isn't this great?!' But it wasn't great." In a pattern that will be familiar to anyone who's worked on large-scale sites "The website came crashing down under load. It had *never* done that before." For Cummings and his crew, the next couple of days were long, as it became an all-hands-on-deck problem. "It was really difficult for us, because it's one of our highest visibility days and our customers were having a terrible experience."

After getting the situation under control, Nordstrom reacted as most organizations do: put ointment on the still-stinging wound. For them, that ointment was spinning up a complete performance-testing environment. Code, it was decreed, now had an additional gauntlet of load tests to face before making its way into production. It was a reasonable first stab at the problem, Cummings recalls, and "was totally catching problems." But the solution quickly spun out of control: "Because we didn't significantly change how we deployed not only our servers, but our code, it took awhile to get that environment up and running. And so other teams wanted their own complete performance environment, just to test their portions of the code. It was all about environments, more environments," Cummings said.

The solution created a big hit to the already-impacted operations team. Cummings notes that other teams often perceived Operations as the bottleneck, even though he had worked through the numbers and could show they weren't. But with the blooming of all of these environments, Cummings knew this process, while well-intentioned, just wasn't going to scale. It was this event (and its solution), only

observable in the crispness of hindsight, that started Nordstrom on its continuous delivery and DevOps journey.

Enter: The "DevOps Team"

That journey started like it often does in larger organizations: with the creation of a "DevOps team," though they didn't name it that. Such teams are topics of hot conversation within the DevOps community —do they work, aren't they just creating another siloed team, how can they possibly help with the necessary cultural changes—but for Nordstrom, getting started on the journey trumped debating the "conventional wisdom" (if such a thing exists in a movement on the heels of celebrating its fifth birthday[1]) about what the "thought leaders" think it has to look like. Doug Ireton, a Nordstrom infrastructure engineer, was one of the first engineers to join this new team. "The first thing we tried to do was pick something small, so we picked host files," which were used extensively within the infrastructure at the time, Ireton said.

The team had settled on Chef as their tool of choice to start implementing "infrastructure as code," one of the pillars of DevOps practice. Server host files seemed like a good idea to Cummings too, who was now leading the team as engineering manager. Working on a heavily-used, production-required element within their infrastructure would give them real-world experience not only learning the automation tool, but also figuring out which workflow was best for them and their own organizational and technological requirements. Plus, it was a small, easy-to-identify scope of work, not too prone to so-called bike-shedding.[2]

"It sounds really good in theory," Cummings said. "Bad idea, it turns out." The pervasive nature of the host files are precisely what made it difficult for the team to pull this one aspect of their entire infrastructure under the control of the new initiative. "When you try to bring in your legacy snowflakes, it's bad. And we're talking hundreds and hundreds of snowflakes, across about twenty environments, and that singular file was different in ways we weren't expecting. That made the implementation extremely complex, just for managing this one file,"

1. DevOps Days Belgium 2014
2. Also known as Parkinson's law of triviality; C. Northcote Parkinson's argued in 1957 that organizations give disproportionate weight to trivial issues; its use was popularized in the software industry in 1999 by the BSD community.

Cummings explains. The idea of trying to shoe-horn a singular, widely-used element of their infrastructure hadn't worked and had frustrated the team to boot.

Try, Try Again

After struggling so much to put something seemingly so simple under configuration management, Cummings realized this approach wasn't going to result in success. He realized the team was still figuring out the fundamentals of not only a new technology, but a new way of modeling and interacting with the systems that comprised their infrastructure. Fortuitously, another project that was critical to the business presented itself: the "payment store processor." Servers at each Nordstrom location ran a legacy application that needed to be virtualized. Experience had shown that creating this server was a manual process that took about 18 hours. At 200 stores, the staggering scope of the task, were they to do it manually, became clear. "We decided to totally pivot our approach and tackle this: we were going to build this server end-to-end, all with Chef," Cummings decided. "But it was Windows Server 2003, so it was the hardest thing you can imagine to automate."

To make the project successful, Cummings roped in engineers from application development, the database team, and other layers of the stack. Despite having a much larger scope and being a more difficult technological problem (and one not even related to the website!), it increased the team's focus. After a few weeks locked in a conference room working together, the team was able to build these "store processor" servers in four hours, in a fully-automated, repeatable fasion. The time-savings to the business and the sanity-savings to the engineers who would be conscripted to do the 18 hours of manual work, in shifts, were obvious. It also gave Cummings' team confidence that the tool really could perform in an odd environment, with a nonstandard use case on an older OS and foreign platform, yet serve a very real business need, and do it end-to-end. "In some ways, this was the hardest thing we could've picked," Ireton notes. "But it really made our team gel."

Reflections on the Journey

The journey is more important than the destination, or so the saying goes. The sentiment could not be truer of an organization's DevOps

transformation. In discussions with engineers on both sides of Nordstrom's technology organization, a number of lessons were highlighted on their path toward an operations environment based on infrastructure as code and development teams taking more ownership and moving toward continuous delivery.

The First Project Is Exactly That: Your First Project

Counter to Apollo 13 Flight Director Gene Kranz's famous quotation, when embarking upon a journey to transform company culture and technological practice, there are a lot of moving parts: failure is an option. As Nordstrom's first "DevOps project" illustrates, it's important to remember that the first attempt at working on a concrete project to incorporate these new ideas into your organization may not result in a completed, fully functional continuous delivery pipeline, backed by your next-generation configuration management tool of choice.

But that doesn't mean the experience is worthless. In the throes of stumbling around, your teams can gain a lot of valuable insight about the intricacies of the technology stack they've chosen, the nuances of the workflows around those tools, and the organization's unique touch points that will be required to make your teams successful. It is also likely to reveal assumptions about your infrastructure that will be sobering to your team, like just how many "server snowflakes" have accumulated to create that snow drift everyone has avoided shoveling.

In the end, it's all about the framing of the initial project and how the team grapples with the outcome, whatever it may be. Despite the initial stumble with managing something "simple," like host files, across the entire infrastructure, Cummings considers the most valuable part of the successful "payment store processor" automation project to be how it kickstarted their journey, even though it wasn't their first attempt: "By having people from those silos all working together, it built a lot of empathy. And we were finally able to get moving."

Change Is a Difficult Process

That change is a difficult process is not a revelation to anyone who's grappled with it in a personal or organizational context. What may be surprising is the specific ways in which the difficulty of change presents itself when working toward adopting a more DevOps-like culture: "You're potentially asking these senior engineers who are experts in a specific ecosystem to move away from that, and sort-of start over,"

Ireston said. That can be a tough sell, especially when a team is already accountable for keeping the business' lights on.

In Nordstrom's case, the experience of looking deliberately at the team's problems and the change required to move them toward continuous delivery involved looking at how teams were structured: "We're trying to make a cultural change to a model where development teams own their app and they run it in prod and they care about it," Ireton said. Nordstrom had previously structured their technology operations around "shared service teams," like QA or operations. To get teams to feel like they actually had ownership over their applications, that meant those previously "shared" roles needed to be embedded, as appropriate, within the application teams. The idea of "shared service" teams also had to shift from the concept of engineers providing a service, like QA running tests, to engineers developing and supporting a service to *provide a service*, such as integration tests or virtual machine deployment, which application teams could then use as they needed. One might even call it a "Service as a Service" model. Change is often also measured, and Nordstrom continues to support the shared services teams for development teams that are still evaluating exactly what a move to an embedded, "full stack" team structure would mean for them.

Another insight Ireton noticed was how certain technology stacks can assist or hinder these sorts of transformations: in Nordstrom's case, the way Microsoft's technology stack interacted with itself tended to be very siloed. Ireton was, in fact, originally hired because of his deep experience with Microsoft's Windows Server Group Policy. "The ecosystem was such that you had the area you knew and were responsible for, but if you needed to go beyond that, you had to find an engineer that was 'Microsoft certified' for that area," Ireton explained. That's a very different model from the "full-stack" teams tasked with direct involvement with all aspects of their application's needs that Nordstrom wanted to move toward.

Prioritization Is the Elephant in the Room

Often, "the business" plays the role of product owner and drives the prioritization of work. But this simplistic model can have disastrous effects if you're trying to introduce an initiative such as continuous delivery. "At the time, for the business, all they wanted was more features," Kissler recalls. "We had to use data showing the sometimes-difficult outcomes, the system outages, and the missed feature

commitments to illustrate that we needed to focus on our technical debt." Kissler said one of the big questions that started being asked was "Why aren't we focusing on these production issues, and why does it take so long to get a feature into production?"

Repeatedly asking these question spurred discussions that allowed the development teams to successfully get a notable portion of each release cycle dedicated to not only paying down technical debt, but specifically for working on developing a continuous delivery pipeline and migrating applications to use it. "After we got someone who had tremendous credibility on the business side who was able to surface those problems in those discussions, it really shifted; then it wasn't a technology story about this whiz-bang continuous delivery thing, it was a story about how the product we were delivering wasn't meeting their needs," Kissler explained. "That story resonated."

Don't Tie the Initiative to Individuals

Once Kissler found her counterpart on the business side, keeping the journey going became easier. But Kissler has a warning about how it could have played out: "You should never create process or confidence around an individual, because that person is not going to be around forever." In Nordstrom's case, her business team counterpart moved, and the initiative stalled. Without someone in the business meetings keeping the torch burning and explaining how the project was doing, the initiative started to backslide, Kissler recalls: "Things got pretty rocky for a bit; I wouldn't say they fell apart, but we certainly had some bumpiness. The organization wanted to return to its previous state."

Interestingly enough, this prompted Kissler and Cummings to actually shift from a story focused on business needs back toward telling a technology story, now that the business had a taste for what was possible. Ultimately, the situation was a mere speedbump on the road of change, but given other circumstances, the departure of key people driving the change can bring an abrupt halt to the journey, sometimes in ways that aren't immediately visible.

Savings Versus Speed

Many enterprises approach IT with the goal of cutting costs wherever possible. This makes sense in a model where the IT department is accounted for as a cost center. But Nordstrom, like many enterprises, realized that isn't the path to the desired results, especially when it

comes to their online presence: "We, as a technology organization, had been optimizing for cost. A couple of years ago, we realized we need to be about speed-to-value, especially in our customer-facing areas," Kissler recalls. "It was challenging to get everyone to make that transition."

As an example, Kissler managed the rollout of the first in-store mobile application to handle sale transactions. The application took six months to develop, and included a lot of features that ended up addressing use cases that customers in stores didn't care about; those features ended up being thrown away. Even the platform—iOS versus Windows tablet—changed. But Kissler wouldn't have done it differently: "In the spirit of speed-to-market, that was our fastest path." After that first successful project, even though its scope was larger than an initial project Kissler would scope and undertake now, it translated into great strides on the journey: "Once we had that, people were like 'let's do more of this.' Let's figure out how we can test and learn, pivot, and fail fast, and create this environment where we can do more of this."

Determine the Flow of Value

One technique that kept coming up in discussions with Nordstrom's application development and operations teams was the process of value stream mapping. Value stream maps attempt to model work that flows through a system. It captures where handoffs occur and how various teams turn raw materials, such as commits, into finished products the business can sell to a customer (or utilize, like an e-commerce website). It is especially good at illustrating the delta between "work as perceived" and "work as performed."[3] Cummings describes a quintessential example of this situation: "Teams would say 'Don't worry about us; our piece is automated.' But then you'd go and look at why things took so long, and the 'automation' was an engineer following a Word document to process something, or some team was undoing work the team before it had done."

Kissler echoes "I need to be able to deliver faster; I want continuous flow with as close to zero waste as possible. So when we needed data to make the case for our business teams, it made it hard for people to

3. A concept described further by Sydner Dekker, as a model used in describing events during postmortem analysis.

dispute it when we surfaced the waste and made it visible." The result of these value flow exercises has revealed so much actionable data that it's still an on-going process for Nordstrom as an organization to work through how to holistically address it: "With the release teams, the development teams, the ops teams, and the QA teams optimizing locally for such a long time, they were hurting the whole system; we're still working through detangling that," Cummings said.

Beware How and Where You Pour Gasoline

When organizations decide to embark upon a journey, they have often committed to make the required investment. But Kissler warns that this investment must be tempered by the organization's ability to absorb the influx of resources, and a systems-view must be taken to ensure the extra resources aren't just being converted to waste: "We tripled our investment in this area, but it caused a problem where we were producing so much, not all the teams could handle it yet. We had to deal with what we called the 'burst-pipe' problem."

This is a common issue where the organization has decided to commit itself to continuous delivery, but the investment is spread unevenly across the organization or teams start to make heavy use of the so-called continuous delivery pipeline while it's still "under construction." Operations and other support teams are used to the metaphor of a life of "fighting fires." When those teams have only ever been given the resources to beat back fires to hidden but still smoldering embers, dumping gasoline on the situation—in the form of increased budget and hiring capability—can cause them to explode into raging fires again. In Nordstrom's case, this required focused investment on the operations side, to reduce time required to build and deploy infrastructure and increase consistency. Had they not undertaken this, the increased investment in development would have hit a major clog in the pipe when it came time to deploy to production. A similar situation applies to the quality assurance part of the pipeline. These clogs can contribute to the "burst pipeline" problem Kissler described.

"There's No Place Like Home"

Oftentimes, part of an organization's cost-cutting strategy involves outsourcing various IT or development functions to other companies. This can be a big impediment to a shift to continuous delivery since the model for outsourced teams typically has them delivering their

artifacts and moving on. This makes it difficult to inculcate a DevOps-focused culture, where teams are responsible for their work via the operation and care-and-feeding of their application. Even though Nordstrom worked with external partners initially on developing some of their new applications and on their configuration management rollouts, they quickly realized they'd need to develop these capabilities internally to really be successful and further leverage that success: "Over time, we said we need to build this in house, or we won't be able to move as fast as we need to," Kissler said. This is not to say that Nordstrom didn't use consultants where it made sense, but they must be employed judiciously: as expert advisors providing guidance, not staff augmentation.

Nordstrom has accomplished this by adding talent, but also a focused investment in cultivating skills for its current employees. It is notable that the Nordstrom employees interviewed for this case study each had their own personal story through various roles and responsibilities at the company, often entailing entirely new skill sets: Cummings started as an operations engineer and moved into program management for infrastructure engineering; Ireton was originally hired for a very specific Windows skill set, but is now developing configuration management infrastructure after having stints on one of the build engineering teams; and Kissler has worked on both sides of the development and operations organization and at various points in time, has owned numerous parts of Nordstrom's in-store and customer mobile strategy.

A "Have-Coffee" Culture

Any discussion surrounding DevOps and its methodologies quickly comes to the often delicate issue of organizational dynamics and culture, at least if it's an accurate treatment of the topic. There is often a tendency to downplay or gloss over these issues precisely because culture is thought of as a "squishy" thing, difficult to shape and change, and in some cases, to even address directly. But it doesn't need to be this way.

Sam Hogenson, Vice President of Technology at Nordstrom, works hard to make sure it's exactly the opposite: "At Nordstrom, we value these different experiences and we value the core of how you work, how you build relationships much more than whether or not you have subject matter expertise. It's a successful formula." Another part of that

formula, Hogenson notes, is the ethos of the organization: "It's a very empowered workforce, a very decentralized organization; I always remember the Nordstroms telling us 'Treat this as if it were your name over the door: how would you run your business and take care of your customers?'" Ireton described it as a "have-coffee culture: if you need to talk to someone, you go have coffee with them."

Planting the Seeds

This mindset has interesting implications when observed in a technology department in the throes of its own transformation. Hogenson describes the complexities of fostering cultural change in a system with a large technological component using the metaphor of a garden: "The biggest job is getting the seeds planted, and the seeds for continuous delivery are planted at Nordstrom; it's getting those seeds from people like Rob Cummings, and then it's a small bit of top-down leadership and committed investment for the garden; then you put down some anti-weed spray to make sure there's space for those seeds to grow, and then you just need to pay very close attention to that part of the garden for awhile, because if you forget to water it or don't tend to the weeds, it will die very quickly."

Hogenson also takes care to make a distinction between a "push model" and a "pull model." Perhaps surprisingly, for all of the development work completed on Nordstrom's continuous delivery pipeline, it's not required that application teams use it. Hogenson notes that the investment in the pipeline is critically important to the company's success, but also knows that some teams may have a "burning platform" that are a higher priority, to both themselves and the business, to get addressed first. "The things that die are the things you try to shove down people's throats," Hogensen said.

"If there's a place that doesn't want to use it right now, that's fine; there's others that will and they'll demonstrate the value. And soon enough, it'll be organic and spread; in our culture, I can't go 'Well, this is the right idea, so you know all that stuff I tell you about ownership and empowerment, well, that doesn't apply to you because I don't agree with you.'" But, creating a "safe space" to cultivate these new ideas and give them some time to become fully formed—this garden, as Hogenson calls it—is critically important to moving the organization forward, and doing so in a way that meshes with the culture the organization professes to believe in.

Speaking the Business' Language

An issue that many organizations struggle with is how to sell it to the business, and Hogenson notes that Nordstrom isn't any different. But it's a problem that he's keenly aware of, and his solution has been to help cultivate another skill in his staff: speaking adeptly to "the business": "What I try to do is really listen and then coach my staff on how to speak to the topic so our business can understand. We sell shoes, and so Rob's gotta articulate in a way that's going to connect those dots, from continuous delivery to shoes, for us." Hogenson notes that both Cummings and Kissler's teams have succeeded at this task: "Our CFO continues to pour money into our technology investments, because our teams have shown they have the credibility to deliver, and because the return on investment is great," Hogenson said. "That has made future conversations a lot easier, too; when done correctly with the right culture, it's a virtuous cycle. Tending that garden early on is paying off."

Nordstrom's deliberate treatment of its corporate culture and self-awareness around how it permeates the decisions it makes and how it affects its various different teams is a component of their success in delivering the right technology to serve its customers. "Continuous Delivery and DevOps, as 'movements', will come and go," Hogenson explains. "What remains? Culture. That's the thing that enables you to realize when it's time to adopt something new *and* and when it's time to move on." Hogensen is surprisingly frank about Nordstrom's odometer reading on the journey: just a few years into a conscious reforming of how its technology teams mesh with themselves and the larger business, he estimates that Nordstrom is only halfway through working to address the discoveries they're unearthing while tilling that garden.

But the process, even though it dirties your hands, Hogenson says, is what makes the other stuff possible: "If you don't pay attention to culture, everything is really hard to do. But if you do, everything else works."

Flipping On the "DevOps Bit"

It would be inaccurate to present a picture that implies Nordstrom's journey is complete, with all of their applications—in-store, on the website, and on mobile—deployed continuously, with developers and operations living together in constant harmony and using a flawless,

infrastructure-as-code-backed pipeline. Every single person interviewed for this case study tempered the stories about the progress that they've made with caveats that there's still more work to be done, more teams to bring aboard. But the successes are undeniable and present themselves in ways both large and small: Cummings recounts a recent "emergency change": "It was a total non-event, because we had infrastructure as code in place." He didn't even have to drive to the colocation facility.

Nordstrom's infrastructure team is currently investing a lot in developer-focused APIs that wrap their core services, like DNS and VM management; they're also working on providing APIs that can unlock the stores of data surrounding their infrastructure, so teams can not only get insight into the running system, but make good decisions for their own applications. There's a nascent public cloud initiative, which seeks to back these APIs with the capability of public clouds—Nordstrom is currently looking at two such providers—in addition to their own internal infrastructure. Application teams will be able to use that API to manipulate and get data from both environments, making the transition easier. Of course, Nordstrom's internal and customer-facing applications continue to be redesigned as teams have bandwidth to pay down technical debt and migrate their builds to a process that fits into the continuous delivery pipeline.

Kissler's goal is to bring the agility of the company's mobile applications to the store application; she knows they may not want to deploy as quickly as the mobile team, but Kissler wants to offer her VP the ability to ship whenever she wants, so it becomes a business decision, unfettered by technology constraints. "They like that story, but they have a hard time believing we can pull it off," so Kissler and her team are walking them through the value stream process that helped their mobile team continuously deliver. Ireton echoes the sentiment with his recent experiences working with various application teams: "I think we're mostly over the hump, at least in how people think and feel about the problem. Not all teams are doing continuous delivery or using our pipeline, but more and more teams want to be."

When asking Nordstrom's team for any advice they'd have given themselves at an earlier pit-stop on their journey if they could, a lot of it surrounds how they'd communicate differently with consumers (i.e., application teams) and bring them into the fold earlier in the process; retrospect also offers pointers on initial projects they may have scoped or even approached differently.

But it was Hogenson who replied quickly and decisively, with the simplest advice: "Keep going."

O'REILLY®

User Story Mapping

DISCOVER THE WHOLE STORY,
BUILD THE RIGHT PRODUCT

Jeff Patton
with Peter Economy

Forewords by Martin Fowler,
Alan Cooper, and Marty Cagan

CHAPTER 3

Plan to Learn Faster

This is my friend Eric, standing in front of his backlog and task board in his team room. He's a product owner working hard with his team to build a successful product, but right now it's not. That doesn't worry Eric, though. He has a strategy for making his product successful. And so far it's working.

Eric works for a company called Liquidnet. Liquidnet is a global trading network for institutional investors. Long before Eric came to stand in front of the board in the picture, someone at his company identified

a group of customers Liquidnet could serve better, along with a few ideas of how to do that. Eric is part of a team that took those ideas and ran with them. That's what product owners do. If you thought they were always acting on their own great ideas, well, you're wrong. One of the hard parts of being a product owner is taking ownership of someone else's idea and helping to make it successful, or proving that it isn't likely to be. The best product owners, like Eric, help their entire team take ownership of the product.

Start by Discussing Your Opportunity

Eric didn't start his work by building a backlog of user stories. He started with the big idea someone had, and treated it like an opportunity for his company—because it was. He had conversations with leadership in his company to understand more. They discussed:

- *What* is the big idea?
- *Who are the customers?* Who are the companies we think would buy the product?
- *Who are the users?* Who are the types of people inside those companies we think would use the product, and what would they be using it for?
- *Why would they want it?* What problems would it solve for customers and users that they couldn't solve today? What benefit would they get from buying and using it?
- *Why are we building it?* If we build this product and it's successful, how does that help us?

Eric needed to build shared understanding with others in his organization before he could take ownership of the opportunity. He knew he was going to need to tell the story of this product many times over the coming months, so he'd better get the big stuff right now.

Your first story discussion is for framing the opportunity.

Validate the Problem

Eric trusts his leadership's intuition, but he knows that this big idea is a hypothesis. He knows the only way to be sure the idea will succeed is when they actually see it succeed.

He first spent time talking to customers and users directly to really learn about them. Along the way he validated that there really were customers who had the problem, and they really were interested in buying a solution. Eric talked to the people who'd likely use the product. They didn't have the product today, and had only poor workarounds to address the problems the new product idea would solve.

Validate that the problems you're solving really exist.

While Eric's been talking with customers and users, he's been building up a pool of people he thinks are good candidates to try his new software. Some companies refer to these people as *customer development partners*. Keep track of this detail, because it's going to come up later in the story.

Actually, during this stage, it wasn't just Eric. Eric was working with a small team of others who spent lots of time talking to their customers, and, in doing so, found that solving the problem wasn't so easy—and that there were other problems that needed to be solved first. The important thing for you to take away is that the more they learned, the more the original opportunity was changed—eventually, a lot. It's lucky they didn't just get to work building what they were told to. That wouldn't have served their customers or their organization.

By now Eric and his team, after talking to customers, had specific ideas for the type of solution they could build that users could use, and by doing so get the benefit their employers wanted. Now, here's where Eric and his team could have gone "all in"—where they could have bet it all. They could have built a backlog of stories that described their solution and set a team to work building it. Because they're smart people, they'd have used a story map to move from the big idea to the specific parts to build. But, because they're *really* smart, the last thing they're going to do at this point is to build software.

Prototype to Learn

It's around here that Eric began to act as the owner for this product. He moved to envision his solution first as a bunch of simple narrative stories—*user scenarios*. Then he moved to envisioning the idea as a simple wireframe sketch. And then he created a higher-fidelity prototype. This wasn't working software. It was a simple electronic prototype, created with a simple tool like Axure, or maybe even PowerPoint.

All of these are learning steps for Eric. They help him envision his solution. Ultimately, he wants to put his solution in front of his users to see what they think. But he knows he first needs to feel confident it solves their problems before he puts it in front of them.

Sketch and prototype so you can envision your solution.

Now, I've hidden an important detail from you. Eric was actually an interaction designer. He's the kind of designer who's used to spending time with customers and users, and used to building these simple prototypes. But, for this new product, he's also the product owner—the one ultimately responsible for the product's success. There are other product owners in Eric's company who don't have his design skills, and they very sensibly pair with designers to help with both interviewing users and envisioning solutions.

Eric did eventually bring prototypes back to users. And I wasn't there, so I don't know what really happened for Eric. But I've been in these situation lots of times, and I'm always surprised about what I learn from the people who'll really use my solution. All I can tell you is, be prepared for surprises and bad news. In fact, celebrate the bad news, because you could have received the same bad news months later, after you'd built the software. That's when it really sucks. Right now, it's cheap to make changes, and you should. And Eric did.

Prototype and test with users to learn whether your solution is valuable and usable.

After iterating his solution many times and showing it to his customers, Eric was confident he had a pretty good solution idea. Surely now he could get that backlog built, and get his team of developers to work

turning that prototyped solution into real working software. But Eric's not going to do that. Well, not exactly that. That's a bigger bet than he's willing to take.

Watch Out for What People Say They Want

Eric has prototyped what he believes is a viable solution. But he's not really sure if it's minimal—because he showed people lots of cool ideas. And if you show people all the cool ideas, of course they'll love them. But Eric knows his job is to minimize the amount he builds and still keep people happy. How much could he take away and still have a viable solution?

Eric also knows something else that's a bit disturbing. He knows the people who said they'd like it and use it are just guessing, too.

Think back to when you've bought something yourself. You may have looked at the product. You might have watched a salesperson demonstrate the cool features. You may have tried out the cool features for yourself, and you could imagine really using and loving the product. But when you bought the product and actually started using it, you found the cool features didn't matter so much. What really mattered were features you hadn't thought about. And, worst of all, maybe you didn't really need the product that much after all. OK, maybe it's just me I'm talking about. But I've got lots of stuff in my garage that I wish I'd never bought.

Back to Eric. He knows his customers and users can imagine the product would be great to use, and knowing that gives him the conviction to up his bet. But the real proof is when those people actually *choose* to use it every day. That's the real outcome he's looking for—and the only outcome that'll get his company the benefit it really wants. And it's going to take more than a prototype to learn that.

Build to Learn

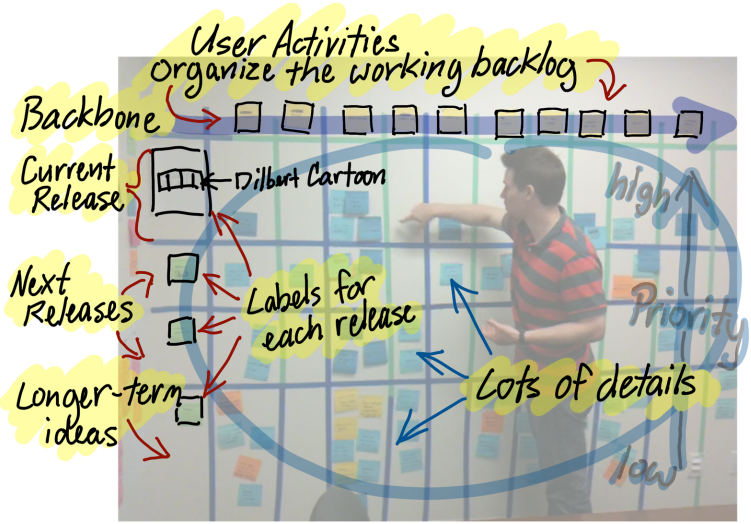
Now, here's where Eric gets to show how smart he really is.

Eric and his team actually do get to work building software. But their first goal isn't to build a minimum viable product. Actually, it's to build something less than minimal—just enough that potential users could do something useful with it. This is a product that wouldn't impress too many people, and they might even hate it. It's definitely not a

product you'd want your marketing and sales people out there pitching. In fact, the only people you'd want to see this product are people who may one day use the product, and honestly care about finding a product that solves their problem.

It just so happens that Eric has a small group of people just like that. It's the customers and users he worked with earlier when he was learning about and validating the problem. They're his development partners. They're the ones who gave feedback on early prototypes. And there's a subset of them that Eric believes can best help him learn. They're the ones he'll put this first, less-than-minimum—and definitely not viable—product in front of. He hopes they'll become his early adopters.

And that's what he did.



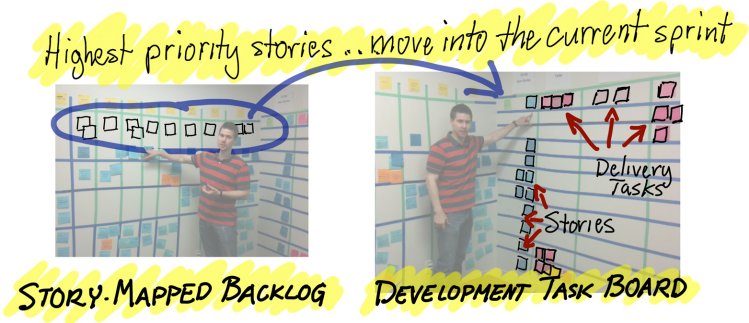
This is Eric pointing out a slice of his current backlog. When this picture was taken, he'd already released software to his development partners. After he did, he made a point of talking to them to get their feedback. His team also built in some simple metrics so they could measure whether people were really using the software, and what they did in the software specifically.

Eric knows that people are polite. They may say they like a product, but then never use it. The "using it" is the real outcome he wants, and polite isn't helping him. Eric also knows some people are demanding.

They may list all the problems the product has, or complain about bugs, but the metrics may be telling us that they use it every day anyway. And that's a good thing, in spite of all their complaining. The complaining is good, too, because it gives Eric ideas about where his next improvements should be.

Eric's backlog is organized as a story map with the backbone in yellow stickies across the top. Those yellow stickies have short verb phrases on them that tell the big story of what his users will do in the product, but at a high level. Below it are all the details—the specific things they'll do and need to really use the product. While the details he and his team work on change from release to release, the backbone stays pretty consistent.

The top slice, above the tapeline, is the one Eric and his team are working on right now. This release will take Eric two sprints. He's using a Scrum development process where his sprints are two-week timeboxes. So two sprints equate to basically a month. Below that are slices running down the board. The next slice contains what they think the next release might be, and so on. To the left of each slice, just as with the Globo.com team, hangs a sticky note with the release name and a few words about what they want to learn in this release. Except for the top release, which has a *Dilbert* cartoon posted over it. It's an inside joke in their team that I wasn't in on.



If you look closely, the top of that current slice is sort of cleaned out. I've drawn in some sticky notes where they used to be. But they're not there anymore because those things that were on the top are the first things his team will build. As the team members worked together to plan their work, they removed those sticky notes and placed them on a task board to the right of the story-mapped backlog. That task board

shows the stories they're working on now in this sprint, along with delivery task—the specific things that the developers and testers will need to do to turn the ideas in the story into working software.

One finer point of Eric's story-mapped backlog, and one that proves he's smart, is the thickness of that topmost slice. It's twice as thick as the slices below it. When Eric and his team finish a slice and deliver it to their development partners—what they call their *beta customers*—they'll move the sticky notes up from the slice below. When they do, they'll have lots more detailed discussion about this next sliced-out release. They'll play What About to find problems and fill in details. They'll talk about some of the ideas in the next release, and this discussion may result in their splitting the big idea into two or three smaller ideas. And then, they'll need the vertical height in that slice to prioritize—to make choices about what to build first.

See how smart they are?

Iterate Until Viable

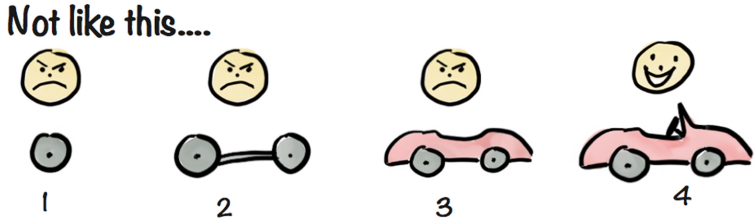
Eric may have started this whole process with an idea about what the minimum viable product might be, but he's purposely built something less than minimal to start with. He's then adding a bit more every month. He's getting feedback from his development partners—both the subjective stuff from talking to them, and the more objective stuff he gets from looking at data.

He'll keep up this strategy, slowly growing and improving the product, until his development partners actually start using the product routinely. In fact, what Eric's hoping for is that they become people who'd recommend the product to other people—real reference customers. When they do, that's when he knows he's found minimum *and* viable. And that's when the product is safe to market and sell like crazy. If Eric and his team had tried to sell it before, they'd have ended up with lots of disappointed customers—people a lot less friendly than those with whom he built personal relationships throughout this process.

How to Do It the Wrong Way

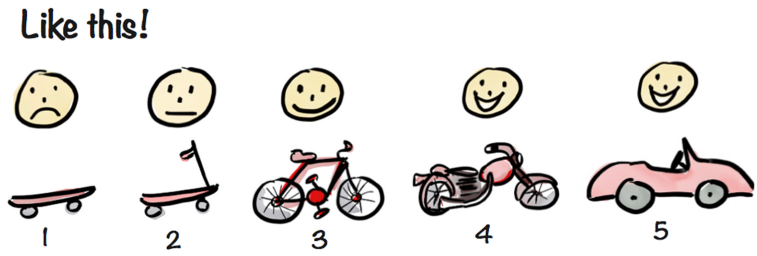
What Eric could have done is to take his last, best prototype, break it down into all its constituent parts, and start building it part by part. Many months later, he'd have had something to release. And he'd have

learned then if his big guess was right. You'll need to trust me on this, but it wouldn't have been—because it rarely is.



This is a simple visualization made by my friend Henrik Kniberg. It beautifully illustrates a broken release strategy where at every release I get something I can't use, until the last release when I get something I can.

Henrik suggests this alternative strategy:



If I plan my releases this way, in each release I deliver something people can actually use. Now, in this silly transportation example, if my goal is to travel a long distance and carry some stuff with me, and you gave me a skateboard, I might feel a bit frustrated. I'd let you know how difficult it was to travel long distances with that thing—although it was fun to goof around with it in the driveway. If your goal was to leave me delighted, you might feel bad about that. But your real goal was to learn, which you did. So that's good. You learned I wanted to travel farther, and if you picked up on it, you also learned I valued having fun.

In Henrik's progression, things start picking up at around the bicycle release because I can actually use it as adequate transportation. And, at about motorcycle level, I can *really* see this working for me—and I'm having fun too. That could be minimum and viable for me. If I

really love the motorcycle thing, maybe my next best step would be a bigger, faster Harley-Davidson, and not a sports car. I'm headed for a midlife crisis right now and that Harley is sounding pretty good. But it's after I try the motorcycle, and we both learn something from that, that we can best make that decision.

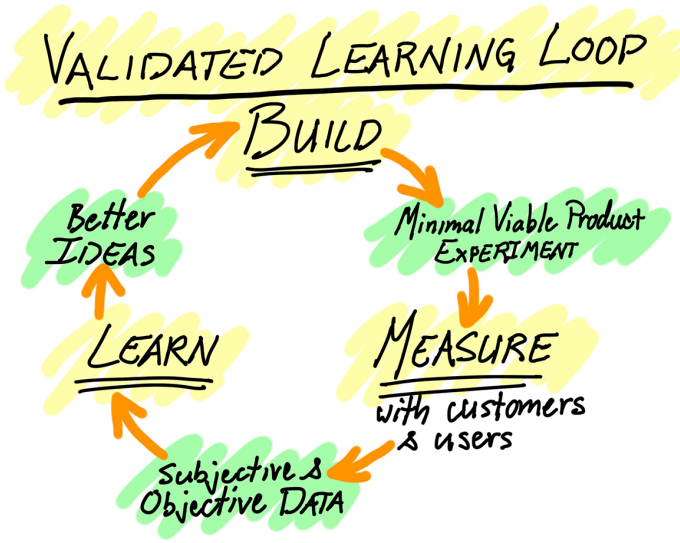
Treat every release as an experiment and be mindful of what you want to learn.

But what about other folks who need to travel longer distance, and who have kids? For *that* target market, none of these would be good choices.

Always keep your target customers, users, and the outcomes you're hoping for in mind. It's really tough to get the same great outcome from all types of users. So focus.

Validated Learning

What my friend Eric did is apply a *validated learning strategy*—one of the important concepts in Lean Startup thinking. Eric knew that the problems he was solving, the customers and users he was solving them for, and the solutions he had in mind were all assumptions. Lots of them were pretty good assumptions. But they were assumptions just the same. Eric set out to understand the assumptions and then validate them, moving from the problems customers and users faced to the solutions he had for them. At each step he did or built something with the explicit goal of learning something.



What Eric did is the heart of the build-measure-learn loop described by Eric Ries. And by Ries's definition, each release that Eric shipped was a minimum viable product. But you can see that it wasn't viable in the eyes of his target customers and users—at least, not yet. For that reason, I like referring to Ries's MVP as a *minimum viable product experiment*—or MVPe for short. It's the smallest thing I could build to learn something. And what I learn is driving toward understanding what's really viable in the eyes of my target customers and users.

Eric used lots of tools and techniques along the way. But telling stories using words and pictures was always part of the way he worked. Using a map to organize his stories helped him keep his customers, users, and their journey in mind as he iteratively improved his product to viable.

I like using the term *product discovery* to describe what we're really doing at this stage. Our goal isn't to get something built; rather, it is to learn if we're building the right thing. It just so happens that building something to put in front of customers is one of the best ways to learn if we're building the right thing. I borrow my definition of *discovery* from Marty Cagan.[1] And my definition of *discovery* includes Lean

1. Marty first described what he means by *product discovery* in this 2007 essay. He later describes it in more detail in his book *Inspired: How to Create Products Customers Love* (SVPG Press).

Startup practice, Lean User Experience practice, Design Thinking practice, and loads of other ideas. And what I do during discovery continues to evolve. But the goal stays the same: to learn as fast as possible whether I'm building the right thing.

Really Minimize Your Experiments

If we recognize that our goal is to learn, then we can minimize what we build and focus on building only what we need to learn. If you're doing this well, it means that what you build early may not be production ready. In fact, if it is, you've likely done too much.

Here's an example: when I was a product owner for a company that built software for large, chain retailers, I knew my products needed to run on a big Oracle database on the backend. But the database guys were sometimes a pain for me to work with. They wanted to scrutinize every change I made. Sometimes simple changes would take a week or more. That slowed down my team and me too much. The database guys' concerns made sense, since all the other applications depended on that database. Breaking it was really risky for everyone. But they had a well-oiled process for evaluating and making database changes —it just took a long time.

The riskiest part for me was making sure my product was right. So we built early versions of software using simple, in-memory databases. Of course, they wouldn't scale, and we could never release our early versions to a large general audience. But our early minimum viable product experiments (we didn't call them that then) allowed us to test ideas with a small subset of customers and still use real data. After several iterations with customers, and after we found a solution we believed would work, we'd then make the database changes and switch our application off the in-memory database. The database guys liked us too, because they knew that when we made changes, we were confident they were the right ones.

Let's Recap

Gary used a map to get out of the flat-backlog trap and see the big picture of his product, and then to really start focusing on who it was for and what it should be.

The teams at Globo.com used a map to coordinate a big plan across multiple teams and slice out a subset of work they believed would be a viable solution.

Eric used a map to slice out less-than-viable releases into minimum viable product experiments that allowed him to iteratively find what would be viable.

There's one last challenge that seems to plague software development, and that's finishing on time. Suppose you're confident that you have something that should be built. And suppose others are depending on it going live on a specific date. There's a secret to finishing on time that's been known by artists for centuries. In the next chapter, we'll learn how to apply it to software.

THE **LEAN** SERIES ERIC RIES, SERIES EDITOR

Jez Humble, Joanne Molesky & Barry O'Reilly

LEAN ENTERPRISE

How High Performance Organizations Innovate at Scale

O'REILLY®

CHAPTER 9

Take an Experimental Approach to Product Development

> The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price the user will pay.
>
> Walter Shewhart

Up to now, we have spent the whole of Part III showing how to improve the speed at which we can deliver value to customers. In this chapter, we switch focus to discuss *alignment*—how to use the capability we have developed to make sure we are building the right things for customers, users, and our organization.

In Chapter 7, we showed how to use the Cost of Delay to prioritize work. In an organization where IT is essentially a service provider, this is an effective way to avoid working on low-value tasks that consume precious time and 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.

In Chapter 6 we described the Improvement Kata, an iterative approach to process improvement in which we set target conditions for the next iteration and then let teams decide what work to do in order to achieve those target conditions. The key innovation we present in this chapter is to use the same process to manage product development. Instead of coming up with

requirements or use cases and putting them into a backlog so that teams build them in priority order, we describe, in measurable terms, the *business outcomes* we want to achieve in the next iteration. It is then up to the teams to discover ideas for features which will achieve these business outcomes, test them, and build those that achieve the desired outcomes. In this way, we harness the skill and ingenuity of the entire organization to come up with ideas for achieving business goals with minimal waste and at maximum pace.

As an approach to running agile software development at scale, this is different from most frameworks. There's no program-level backlog; instead, teams create and manage their own backlogs and are responsible for collaborating to achieve business goals. These goals are defined in terms of target conditions at the program level and regularly updated as part of the Improvement Kata process (see Chapter 6). Thus the responsibility for achieving business goals is pushed down to teams, and teams focus on business outcomes rather than measures such as the number of stories completed (team velocity), lines of code written, or hours worked. Indeed, the goal is to *minimize* output while maximizing outcomes: the fewer lines of code we write and hours we work to achieve our desired business goals, the better. Enormous, overly complex systems and burned-out staff are symptoms of focusing on output rather than outcomes.

One thing we don't do in this chapter (or indeed this book) is prescribe what processes teams should use to manage their work. Teams can—and should—be free to choose whatever methods and processes work best for them. Indeed, in the HP FutureSmart program, different teams successfully used different methodologies and there was no attempt to impose a "standard" process or methodology across the teams. What is important is that the teams are able to work together effectively to achieve the target conditions.

Therefore, we don't present standard agile methods such as XP, Scrum, or alternatives like Kanban. There are several excellent books that cover these methods in great detail, such as David Anderson's *Kanban: Successful Evolutionary Change for Your Technology Business*,[1] Kenneth S. Rubin's *Essential Scrum: A Practical Guide to the Most Popular Agile Process* (Addison-Wesley), and Mitch Lacey's *The Scrum Field Guide: Practical Advice for Your First Year* (Addison-Wesley). Instead, we discuss how teams can collaborate to define approaches to achieve target conditions, then design experiments to test their assumptions.

The techniques described in this chapter require a high level of trust between different parts of the organization involved in the product development value

1 [anderson]

stream, as well as between leaders, managers, and those who report to them. They also require high-performance teams and short lead times. Thus, unless these foundations (described in previous chapters in this part) are in place, implementing these techniques will not produce the value they are capable of.

Using Impact Mapping to Create Hypotheses for the Next Iteration

The outcome of the Improvement Kata's iteration planning process (described in Chapter 6) is a list of measurable target conditions we wish to achieve over the next iteration, describing the *intent* of what we are trying to achieve and following the Principle of Mission (see Chapter 1). In this chapter, we describe how to use the same process to drive product development. We achieve this by creating target conditions based on customer and organizational outcomes as part of our iteration planning process, in addition to process improvement target conditions. This enables us to use program-level continuous improvement for product development too, by adopting a *goal-oriented* approach to requirements engineering.

Our product development target conditions describe customer or business goals we wish to achieve, which are driven by our product strategy. Examples include increasing revenue per user, targeting a new market segment, solving a given problem experienced by a particular persona, increasing the performance of our system, or reducing transaction cost. However, we do not propose solutions to achieve these goals or write stories or features (especially not "epics") at the program level. Rather, it is up to the teams within the program to decide how they will achieve these goals. This is critical to achieving high performance at scale, for two reasons:

- The initial solutions we come up with are unlikely to be the best. Better solutions are discovered by creating, testing, and refining multiple options to discover what best solves the problem at hand.
- Organizations can only move fast at scale when the people building the solutions have a deep understanding of both user needs and business strategy and come up with their own ideas.

A program-level backlog is not an effective way to drive these behaviors—it just reflects the almost irresistible human tendency to specify "the means of doing something, rather than the result we want."[2]

2 [gilb-88], p. 23.

TIP

Getting to Target Conditions

Goal-oriented requirements engineering has been in use for decades,[3] but most people are still used to defining work in terms of features and benefits rather than measurable business and customer outcomes. The features-and-benefits approach plays to our natural bias towards coming up with solutions, and we have to think harder to specify the attributes that an acceptable solution will have instead.

If you have features and benefits and you want to get to target conditions, one simple approach is to ask *why* our customers care about a particular benefit. You may need to ask "why" several times to get to something that looks like a real target condition.[4] It's also essential to ensure that target conditions have *measurable* acceptance criteria, as shown in Figure 9-1.

Gojko Adzic presents a technique called *impact mapping* to break down high-level business goals at the program level into testable hypotheses. Adzic describes an impact map as "a visualization of scope and underlying assumptions, created collaboratively by a cross-functional group of stakeholders. It is a mind-map grown during a discussion facilitated by answering the following questions: 1. Why? 2. Who? 3. How? 4. What?"[5] An example of an impact map is shown in Figure 9-1.

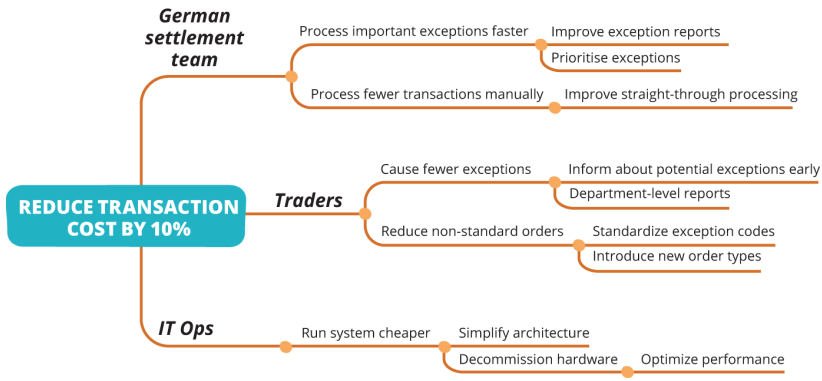


Figure 9-1. *An example of an impact map, courtesy of Gojko Adzic*

We begin an impact map with a program-level target condition. By stating a target condition, including the *intent* of the condition (why we care about it

3 See [yu], [lapouchnian], and [gilb-05] for more on goal-oriented requirements engineering.

4 This is an old trick used by Taiichi Ohno, called "the five whys."

5 [adzic], l. 146.

from a business perspective), we make sure everyone working towards the goal understands the purpose of what they are doing, following the Principle of Mission. We also provide clear acceptance criteria so we can determine when we have reached the target condition.

The first level of an impact map enumerates all the stakeholders with an interest in that target condition. This includes not only the end users who will be affected by the work, but also people within the organization who will be involved or impacted, or can influence the progress of the work—either positively or negatively.

The second level of an impact map describes possible ways the stakeholders can help—or hinder—achieving the target condition. These changes of behavior are the *impacts* we aim to create.

So far, we should have said nothing about possible solutions to move us towards our target condition. It is only at the third level of the impact map that we propose options to achieve the target condition. At first, we should propose solutions that don't involve writing code—such as marketing activities or simplifying business processes. Software development should always be a last resort, because of the cost and complexity of building and maintaining software.

The possible solutions proposed in the impact map are *not* the key deliverable. Coming up with possible solutions simply helps us refine our thinking about the goal and stakeholders. The solutions we come up with at this stage are unlikely to be the best—we expect, rather, that the people working to deliver the outcomes will come up with better options and evaluate them to determine which ones will best achieve our target condition. The impact map can be considered a set of assumptions—for example, in Figure 9-1, we assume that standardizing exception codes will reduce nonstandard orders, which will reduce the cost of processing nonstandard transactions.

For this tool to work effectively, it's critical to have the right people involved in the impact-mapping exercise. It might be a small, cross-functional team including business stakeholders, technical staff, designers, QA (where applicable), IT operations, and support. If the exercise is conducted purely by business stakeholders, they will miss the opportunity to examine the assumptions behind the target conditions and to get ideas from the designers and engineers who are closest to the problem. One of the most important goals of impact mapping is to create a shared understanding between stakeholders, so not involving them dooms it to irrelevance.

Once we have a prioritized list of target conditions and impact maps created collaboratively by technical and business people, it is up to the teams to determine the shortest possible path to the target condition.

This tool differs in important ways from many standard approaches to thinking about requirements. Here are some of the important differences and the motivations behind them:

There are no lists of features at the program level
: Features are simply a mechanism for achieving the goal. To paraphrase Adzic, *if achieving the target condition with a completely different set of features than we envisaged won't count as success, we have chosen the wrong target condition*. Specifying target conditions rather than features allows us to rapidly respond to changes in our environment and to the information we gather from stakeholders as we work towards the target condition. It prevents "feature churn" during the iteration. Most importantly, it is the most effective way to make use of the talents of those who work for us; this motivates them by giving them an opportunity to pursue mastery, autonomy, and purpose.

There is no detailed estimation
: We aim for a list of target conditions that is a stretch goal—in other words, if *all* our assumptions are good and *all* our bets pay off, we think it would be possible to achieve them. However, this rarely happens, which means we may not achieve some of the lower-priority target conditions. If we are regularly achieving much less, we need to rebalance our target conditions in favor of process improvement goals. Keeping the iterations short —2–4 weeks initially—enables us to adjust the target conditions in response to what we discover during the iteration. This allows us to quickly detect if we are on a wrong path and try a different approach before we overinvest in the wrong things.

There are no "architectural epics"
: The people doing the work should have complete freedom to do whatever improvement work they like (including architectural changes, automation, and refactoring) to best achieve the target conditions. If we want to drive out particular *goals* which will require architectural work, such as compliance or improved performance, we specify these in our target conditions.

Performing User Research

Impact mapping provides us with a number of possible solutions and a set of assumptions for each candidate solution. Our task is to find the shortest path to the target condition. We select the one that seems shortest, and validate the solution—along with the assumptions it makes—to see if it really is capable of delivering the expected value (as we have seen, features often fail to deliver the expected value). There are multiple ways to validate our assumptions.

First, we create a *hypothesis* based on our assumption. In *Lean UX*, Josh Seiden and Jeff Gothelf suggest the template shown in Figure 9-2 to use as a starting point for capturing hypotheses.[6]

> We believe that
> [building this feature]
> [for these people]
> will achieve [this outcome].
> We will know we are successful when we see
> [this signal from the market].

Figure 9-2. *Jeff Gothelf's template for hypothesis-driven development*

In this format, we describe the parameters of the experiment we will perform to test the value of the proposed feature. The *outcome* describes the target condition we aim to achieve.

As with the agile story format, we summarize the *work* (for example, the feature we want to build or the business process change we want to make) in a few words to allow us to recall the conversation we had about it as a team. We also specify the *persona* whose behavior we will measure when running the experiment. Finally, we specify the signal we will measure in the experiment. In online controlled experiments, discussed in the next section, this is known as the *overall evaluation criterion* for the experiment.

Once we have a hypothesis, we can start to design an experiment. This is a cross-functional activity that requires collaboration between design, development, testing, techops, and analysis specialists, supported by subject matter experts where applicable. Our goal is to *minimize* the amount of work we must perform to gather a sufficient amount of data to validate or falsify the assumptions of our hypothesis. There are multiple types of user research we can perform to test our hypothesis, as shown in Figure 9-3.[7] For more on different types of user research, read *UX for Lean Startups* (O'Reilly) by Laura Klein.

---

6 [gothelf], p. 23.

7 This diagram was developed by Janice Fraser; see *http://slidesha.re/1v715bL*.

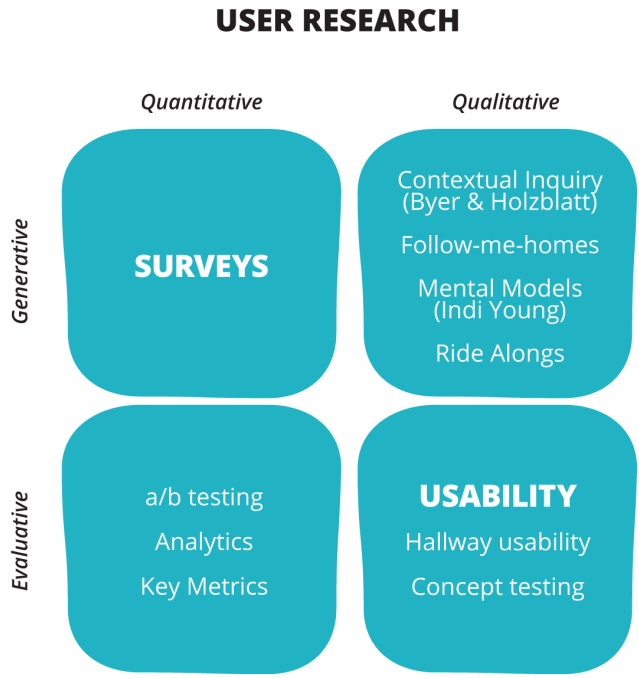


Figure 9-3. *Different types of user research, courtesy of Janice Fraser*

The key outcome of an experiment is *information*: we aim to reduce the uncertainty as to whether the proposed work will achieve the target condition. There are many different ways we can run experiments to gather information. Bear in mind that experiments will often have a negative or inconclusive result, especially in conditions of uncertainty; this means we'll often need to tune, refine, and evolve our hypotheses or come up with a new experiment to test them.

The key to the experimental approach to product development is that *we do no major new development work without first creating a hypothesis* so we can determine if our work will deliver the expected value.[8]

Online Controlled Experiments

In the case of an internet-based service, we can use a powerful method called an *online controlled experiment*, or A/B test, to test a hypothesis. An A/B test

8 In many ways, this approach is just an extension of test-driven development. Chris Matts came up with a similar idea he calls *feature injection*.

is a randomized, controlled experiment to discover which of two possible versions of a web page produces better outcome. When running an A/B test, we prepare two versions of a page: a control (typically the existing version of the page) and a new treatment we want to test. When a user first visits our website, the system decides which experiments that user will be a subject for, and for each experiment chooses at random whether they will view the control (A) or the treatment (B). We instrument as much of the user's interaction with the system as possible to detect any differences in behavior between the control and the treatment.

Most Good Ideas Actually Deliver Zero or Negative Value

Perhaps the most eye-opening result of A/B testing is how many apparently great ideas do not improve value, and how utterly impossible it is to distinguish the lemons in advance. As discussed in Chapter 2, data gathered from A/B tests by Ronny Kohavi, who directed Amazon's Data Mining and Personalization group before joining Microsoft as General Manager of its Experimentation Platform, reveal that 60%–90% of ideas *do not improve the metric they were intended to improve*.

Thus if we're not running experiments to test the value of new ideas before completely developing them, the chances are that about 2/3 of the work we are doing is of either *zero* or *negative* value to our customers—and certainly of negative value to our organization, since this work costs us in three ways. In addition to the cost of developing the features, there is an opportunity cost associated with more valuable work we could have done instead, and the cost of the new complexity they add to our systems (which manifests itself as the cost of maintaining the code, a drag on the rate at which we can develop new functionality, and often, reduced operational stability and performance).

Despite these terrible odds, many organizations have found it hard to embrace running experiments to measure the value of new features or products. Some designers and editors feel that it challenges their expertise. Executives worry that it threatens their job as decision makers and that they may lose control over the decisions.

Kohavi, who coined the term "HiPPO," says his job is "to tell clients that their new baby is ugly," and carries around toy rubber hippos to give to these people to help lighten the mood and remind them that most "good" ideas aren't, and that it's impossible to tell in the absence of data which ones will be lemons.

By running the experiment with a large enough number of users, we aim to gather enough data to demonstrate a statistically significant difference between A and B for the business metric we care about, known as the overall evaluation criterion, or OEC (compare the One Metric That Matters from Chapter 4). Kohavi suggests optimizing for and measuring *customer lifetime value* rather than short-term revenue. For a site such as Bing, he recommends using a weighted sum of factors such as time on site per month and visit frequency per

user, with the aim being to improve the overall customer experience and get them to return.

Unlike data mining, which can only discover correlations, A/B testing has the power to show a *causal relationship* between a change on a web page and a corresponding change in the metric we care about. Companies such as Amazon and Microsoft typically run hundreds of experiments in production at any one time and test every new feature using this method before rolling it out. Every visitor to Bing, Microsoft's web search service, will be participating in about 15 experiments at a time.[9]

Using A/B Testing to Calculate the Cost of Delay for Performance Improvements

At Microsoft, Ronny Kohavi's team wanted to calculate the impact of improving the performance of Bing searches. They did it by running an A/B test in which they introduced an artificial server delay for users who saw the "B" version. They were able to calculate a dollar amount for the revenue impact of performance improvements, discovering that "an engineer that improves server performance by 10 msec more than pays for his fully-loaded annual costs." This calculation can be used to determine the cost of delay for performance improvements.

When we create an experiment to use as part of A/B testing, we aim to do much less work than it would take to fully implement the feature under consideration. We can calculate the maximum amount we should spend on an experiment by determining the expected value of the information we will gain from running it, as discussed in Chapter 3 (although we will typically spend much less than this).

In the context of a website, here are some ways to reduce the cost of an experiment:

Use the 80/20 rule and don't worry about corner cases
: Build the 20% of functionality that will deliver 80% of the expected benefit.

Don't build for scale
: Experiments on a busy website are usually only seen by a tiny percentage of users.

9 *http://www.infoq.com/presentations/controlled-experiments*

Don't bother with cross-browser compatibility
: With some simple filtering code, you can ensure that only users with the correct browser get to see the experiment.

Don't bother with significant test coverage
: You can add test coverage later if the feature is validated. Good monitoring is much more important when developing an experimentation platform.

An A/B Test Example

Etsy is a website where people can sell handcrafted goods. Etsy uses A/B testing to validate all major new product ideas. In one example, a product owner noticed that searching for a particular type of item on somebody's storefront comes up with zero results, and wanted to find out if a feature that shows similar items from somebody else's storefront would increase revenue. To test the hypothesis, the team created a very simple implementation of the feature. They used a configuration file to determine what percentage of users will see the experiment.

Users hitting the page on which the experiment is running will be randomly allocated either to a control group or to the group that sees the experiment, based on the weighting in the configuration file. Risky experiments will only be seen by a very small percentage of users. Once a user is allocated to a bucket, they stay there across visits so the site has a consistent appearance to them.

Making It Safe to Fail

A/B testing allows teams to define the constraints, limits, or thresholds to create a safe-to-fail experiment. The team can define the control limit of a key metric before testing so they can roll back or abort the test if this limit is reached (e.g., conversion drops below a set figure). Determining, sharing, and agreeing upon these limits with all stakeholders before conducting the experiment will establish the boundaries within which the team can experiment safely.

Users' subsequent behavior is then tracked and measured as a cohort—for example, we might want to see how many then make it to the payment page. Etsy has a tool, shown in Figure 9-4, which measures the difference in behavior for various endpoints and indicates when it has reached statistical significance at a 95% confidence interval. For example, for "site—page count," the bolded "+0.26%" indicates the experiment produces a statistically significant 0.26% improvement over the control. Experiments typically have to run for a few days to produce statistically significant data.

Generating a change of more than a few percent in a business metric is rare, and can usually be ascribed to Twyman's Law: "If a statistic looks interesting or unusual it is probably wrong."

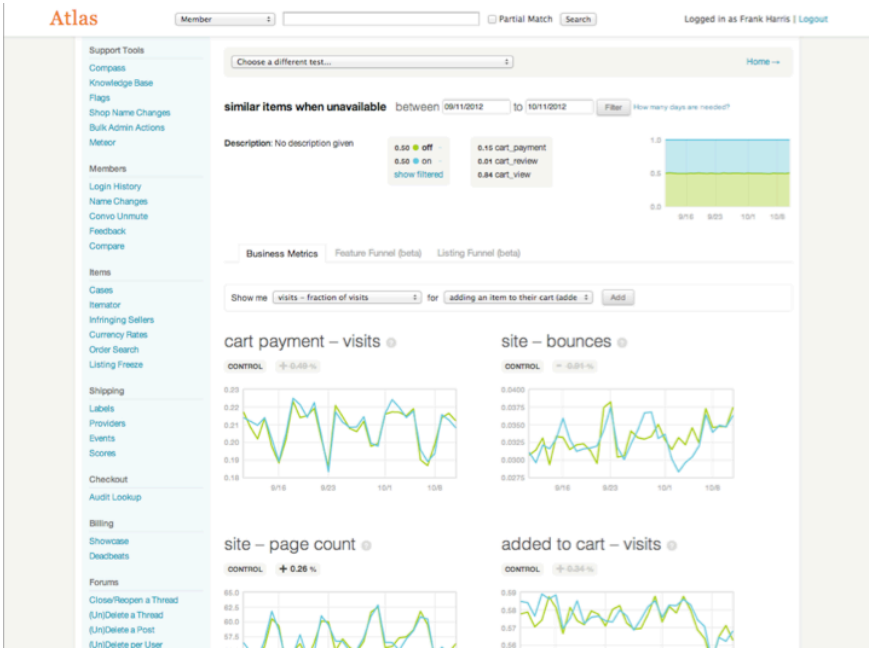


Figure 9-4. *Measuring changes in user behavior using A/B testing*

If the hypothesis is validated, more work can be done to build out the feature and make it scale, until ultimately the feature is made available to all users of the site. Turning the visibility to 100% of users is equivalent to publicly releasing the feature—an important illustration of the difference between deployment and release which we discussed in Chapter 8. Etsy always has a number of experiments running in production at any time. From a dashboard, you can see which experiments are planned, which are running, and which are completed, which allows people to dive into the current metrics for each experiment, as shown in Figure 9-5.

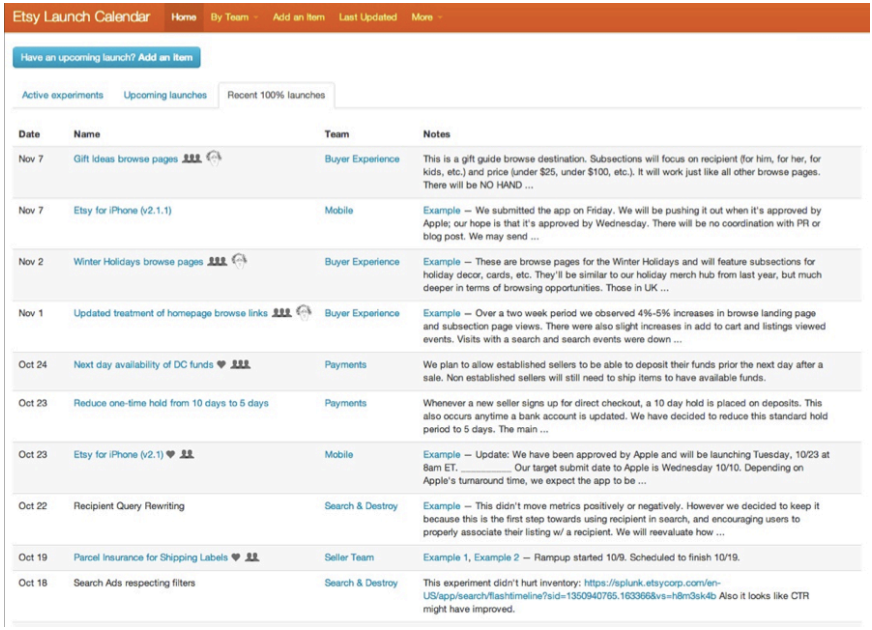


Figure 9-5. *Experiments currently running at Etsy*

Alternatives to A/B Testing

Although we spend a lot of time on A/B testing in this chapter, it is just one of a wide range of experimental techniques for gathering data. User experience designers have a variety of tools to get feedback from users, from lo-fi prototypes to ethnographic research methods such as contextual enquiry, as shown in Figure 9-3. *Lean UX: Applying Lean Principles to Improve User Experience* discusses a number of these tools and how to apply them in the context of hypothesis-driven development.[10]

Prerequisites for an Experimental Approach to Product Development

Convincing people to gather—and then pay attention to—real data from experimentation, such as A/B testing, is hard enough. But an experimental, scientific approach to creating customer value has implications for the way we do work, as well as for the way we think about value. As Dan McKinley of Etsy

10 [gothelf]

points out,[11] experimentation can't be bolted on to a waterfall product development process. If we get to the end of several weeks (or months) of work and attempt an experiment, there's a very good chance we'll find the huge batch of work we did either has zero effect or makes things worse. At that point we'll have to throw it all away because there is no way to accurately identify the effect of each specific change introduced.

This is an extremely painful decision, and in practice many teams succumb to the sunk cost fallacy by giving undue weight to the investment made to date when taking this decision. They ignore the data and deploy the product as is because shelving the work is considered a total failure, whereas successful deployment of anything into production is perceived as success—so long as it is on time and on budget.

If we're going to adopt a thorough experimental approach, we need to change what we consider to be the outcome of our work: not just validated ideas but the information we gain in the course of running the experiments. We also need to change the way we think about developing new ideas; in particular, it's essential to work in small batches and test every assumption behind the idea we are validating. This, in turn, requires that we implement continuous delivery, as described in Chapter 8.

Working in small batches creates *flow*—a key element of Lean Thinking. But small batches are hard to achieve, for both philosophical and technical reasons. Some people have a problem with taking an incremental approach to creating products. A common objection to an experimental approach is that it leads to locally optimal but globally suboptimal decisions, and that it compromises the overall integrity of the product, murdering a beautiful holistic vision by a thousand A/B tests.

While it is certainly possible to end up with an ugly, overcomplex product when teams fail to take a holistic approach to user experience, this is not an inevitable outcome of A/B testing. Experimentation isn't supposed to replace having a vision for your product. Rather, it enables you to evolve your strategy and vision rapidly in response to real data from customers using your products in their environment. A/B testing will not be effective in the absence of a vision and strategy. Product managers, designers, and engineers need to collaborate and apply the lessons of design thinking in order to take a long-term view of the needs of users and establish a direction for the product.

11 *http://slidesha.re/1v71gUs*

TIP

What Is Design Thinking?

Tim Brown, CEO and President of IDEO and one of the key figures in design thinking, says, "As a style of thinking, it is generally considered the ability to combine empathy for the context of a problem, creativity in the generation of insights and solutions, and rationality to analyze and fit solutions to the context." We discuss design thinking and Lean UX further in Chapter 4.

There are two further obstacles to taking an experimental approach to product development. First, designing experiments is tricky: we have to prevent them from interfering with each other, apply alerts to detect anomalies, and design them to produce valid results. At the same time, we want to minimize the amount of work we must do to gather statistically significant data.

Finally, taking a scientific approach to customer and product development requires intensive collaboration between product, design, and technical people throughout the lifecycle of every product. This is a big cultural change for many enterprises where technical staff do not generally contribute to the overall design process.

These obstacles are the reason why we strongly discourage people from adopting the tools discussed in this chapter without first putting in place the foundations described in the earlier chapters in Part III.

Innovation Requires a Culture of Experimentation

Greg Linden, who developed Amazon's first recommendations engine, came up with a hypothesis that showing personalized recommendations at checkout time might convince people to make impulse buys—similar to the rack at the checkout lane in a grocery store but compiled personally for each customer by an algorithm. However, a senior vice-president who saw Greg's demo was convinced it would distract people from checking out. Greg was forbidden to do any further work on the feature.[12]

Linden disobeyed the SVP and put an A/B test into production. The A/B test demonstrated such a clear increase in revenue when people received personalized recommendations at check-out that the feature was built out and launched with some urgency.

Is it even conceivable that an engineer at your company could push an A/B test into production in the face of censure by a senior executive? If the experiment's data proved the executive wrong, how likely is it that the feature would be picked up rather than buried? As Linden writes, "Creativity must flow from everywhere. Whether you are a summer intern or the CTO, any good idea must be able to seek an objective test,

12 *http://bit.ly/1v71kmW*

preferably a test that exposes the idea to real customers. Everyone must be able to experiment, learn, and iterate. Position, obedience, and tradition should hold no power. For innovation to flourish, measurement must rule."

A culture based on measurement and experimentation is not antithetical to crazy ideas, divergent thinking, and abductive reasoning. Rather, it gives people license to pursue their crazy ideas—by making it easy to gather real data to back up the good crazy and reject the bad crazy. Without the ability to run cheap, safe-to-fail experiments, such ideas are typically trampled by a passing HiPPO or by the mediocrity of decision-by-committee.

One of the most common challenges encountered in software development is the focus of teams, product managers, and organizations on managing cost rather than value. This typically manifests itself in undue effort spent on zero-value-add activities such as detailed upfront analysis, estimation, scope management, and backlog grooming. These symptoms are the result of focusing on maximizing utilization (keeping our expensive people busy) and output (measuring their work product)—instead of focusing on outcomes, minimizing the output required to achieve them, and reducing lead times to get fast feedback on our decisions.

Conclusion

Most ideas—even apparently good ones—deliver zero or negative value to users. By focusing on the *outcomes* we wish to achieve, rather than solutions and features, we can separate *what* we are trying to do from the possible ways to do it. Then, following the Principle of Mission, teams can perform user research (including low-risk, safe-to-fail online experiments) to determine what will actually provide value to customers—and to our organization.

By combining impact mapping and user research with the Improvement Kata framework presented in Chapter 6, we can scale agile software delivery and combine it with design thinking and an experimental approach to product development. This allows us to rapidly discover, develop, and deliver high-value, high-quality solutions to users at scale, harnessing the skill and ingenuity of everybody in the organization.

Questions for readers:

- What happens at your organization when a substantial amount of effort has been invested in an idea that turns out to provide little value to users or the organization, or even to make things worse?
- Have the expected customer outcomes for the features you are working on been quantified? Do you have a way to measure the actual outcomes?

- What kind of user research do you perform on prototypes before releasing them more widely? How might you get that feedback more quickly and cheaply?
- When was the last time you personally observed your product used or discussed in real life?
- Can you think of a cheap way to test the value of the next piece of work in your backlog?

THE **LEAN** SERIES

Jeff Gothelf with Josh Seiden

LEAN UX

Applying Lean Principles to Improve User Experience

O'REILLY®

Eric Ries, Series Editor

CHAPTER 7

Integrating Lean UX and Agile

Agile methods are now mainstream. At the same time, thanks to the huge success of products such as the Kindle and the iPhone, so is user experience design. But making Agile work with UX has long been a challenge. In this chapter, I review how Lean UX methods can fit within the most popular flavor of Agile—the Scrum process—and discuss how blending Lean UX and Agile can create a more productive team and a more collaborative process. I'll cover:

Definition of terms

Just to make sure we're all on the same page about certain words like "sprint" and "story."

Staggered sprints

The one-time savior of Agile/UX integration is now just a stepping stone to true team cohesion.

Listening to Scrum's rhythms

The meeting cadences of Scrum are clear guideposts for Lean UX integration.

Participation

A truly cross-functional process requires that everyone be a part of it.

Design as a team sport

Ensuring that the once-closed design process is now open to all team members is key to your success.

Managing up and out

Clear obstacles to your team's progress by being proactive with your communication.

Some Definitions

Agile processes, including Scrum, use many proprietary terms. Over time, many of these terms have taken on a life of their own. To ensure that I'm using them clearly, I've taken the time to define a few of them here (If you're familiar with Scrum, you can skip this section.)

Scrum

An agile methodology promoting time-boxed cycles, team self-organization, and high team accountability. Scrum is the most popular form of Agile.

User story

The smallest unit of work expressed as a benefit to the end user. Typical user stories are written using the following syntax:

As a **[user type]**

I want to **[accomplish something]**

So that **[some benefit happens]**

Backlog

A prioritized list of user stories. The backlog is the most powerful project management tool in Agile. It is through the active grooming of the backlog that the team manages their daily workload and refocuses their efforts based on incoming knowledge. It is how the team stays agile.

Sprint:

A single team cycle. The goal of each sprint is to deliver working software. Most Scrum teams work in two-week sprints.

Stand-up:

A daily short team meeting at which each member addresses the day's challenges—one of Scrum's self-accountability tools. Each member must declare to all teammates, every day, what he or she is doing and what's getting in his or her way.

Retrospective

A meeting at the end of each sprint that takes an honest look at what went well, what went poorly, and how the team will try to improve the process in the next sprint. Your process is as iterative as your

product. Retrospectives give your team the chance to optimize your process with every sprint.

Iteration planning meeting

A meeting at the beginning of each sprint at which the team plans the upcoming sprint. Sometimes this meeting includes estimation and story gathering. This is the meeting that determines the initial prioritization of the backlog.

Beyond Staggered Sprints

In May 2007, Desiree Sy and Lynn Miller published "Adapting Usability Investigations for Agile User-centered Design" in the *Journal of Usability Studies* (*http://www.upassoc.org/upa_publications/jus/2007may/agile-ucd.pdf*). Sy and Miller were some of the first people to try to combine Agile and UX, and many of us were excited by the solutions they were proposing. In the article, Sy and Miller describe in detail their idea of a productive integration of Agile and user-centered design. They use a technique called Cycle 0 (you may have heard it called "Sprint Zero" or "Staggered Sprints" as well).

In short, Sy and Miller describe a process in which design activity takes place one sprint ahead of development (Figure 7-1). Work is designed and validated during the "design sprint" and then passed off into the development stream to be implemented during the development sprint.

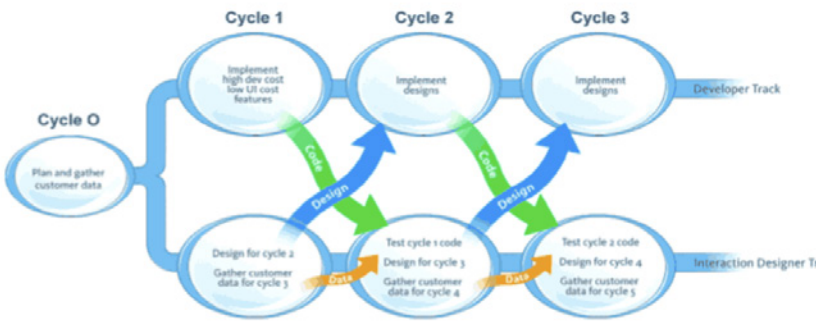


Figure 7-1. *Sy and Miller's "staggered sprints" model.*

Many teams have misinterpreted this model. Sy and Miller always advocated strong collaboration between designers and developers during both the design and development sprints. Many teams have missed this critical point and have instead created workflows in which designers and developers communicate by handoff, creating a kind of mini-waterfall process.

Staggered sprints can work well for some teams. If your development environment does not allow for frequent releases (for example, you work on packaged software, or embedded software, or deliver software to an environment in which continuous deployment is difficult or impossible), the premium on getting the design right is higher. In these cases, Lean UX may not be a great fit for your team, as you'll have to work hard to get the market feedback you need to make many of these techniques work.

For these teams, staggered sprints can allow for more validation of design work—provided that you are still working in a very collaborative manner. And teams transitioning from Waterfall to Agile can benefit from working this way as well, because it teaches you to work in shorter cycles and to divide your work into sequential pieces.

However, this model works best as a transition. It is not where you want your team to end up. Here's why: it becomes very easy to create a situation in which the entire team is never working on the same thing at the same time. You never realize the benefits of cross-functional collaboration because the different disciplines are focused on different things. Without that collaboration, you don't build shared understanding, so you end up relying heavily on documentation and handoffs for communication.

There's another reason this process is less than ideal: it can create unnecessary waste. You waste time creating documentation to describe what happened during the design sprints. And if developers haven't participated in the design sprint, they haven't had a chance to assess the work for feasibility or scope. That conversation doesn't happen until handoff. Can they actually build the specified designs in the next two weeks? If not, the work that went into designing those elements is waste.

Building Lean UX into the Rhythm of Scrum

As I said in the opening to this chapter, we tried using Staggered Sprints at TheLadders. And when we had problems, we continued to improve our process, eventually ending up with a deeply collaborative routine that played out across the rhythms of Scrum. Let's take a look at how you can use Scrum's meeting structure and Lean UX to build an efficient process.

Themes

Scrum has a lot of meetings. Many people frown on meetings, but if you use them as mileposts during your sprint, you can create an integrated Lean UX and Agile process in which the entire team is working on the same thing at the same time.

Let's take a two-week sprint model and assume that we can tie a series of these sprints together under one umbrella, which we'll call a theme (Figure 7-2).

Figure 7-2. *Sprints tied together with a theme.*

Kickoff Sessions for Sketching and Ideation

Each theme should be kicked off with a series of brainstorming and validation exercises like the ones described in Section II. These activities can be as short as an afternoon or as long as a week. You can do them with your immediate team or with a broader group. The scope of the theme will determine how much participation and time these kickoff activities require. The point of this kickoff is to get the entire team sketching and ideating together, creating a backlog of ideas to test and learn from.

Once you've started your sprints, these ideas will be tested, validated, and developed new knowledge will come in, and you'll need to decide what to do with it. You make these decisions by running subsequent shorter brainstorming sessions that take place before each new sprint begins, which allows the team to use the latest insight to create the backlog for the next sprint. See Figure 7-3.

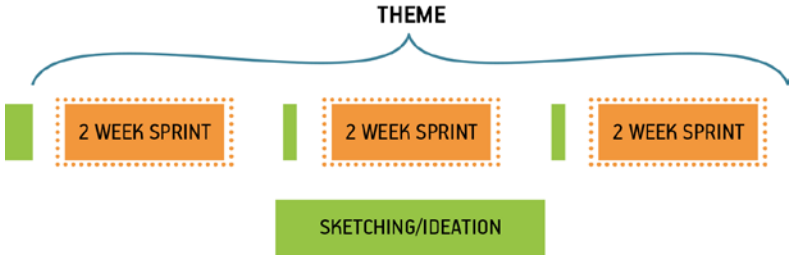


Figure 7-3. *Timing and scope of sketching, ideation, and brainstorming sessions.*

Iteration Planning Meeting

The output of the kickoff session should be brought to the *iteration planning meeting* (IPM). Your mess of sticky notes, sketches, wireframes, paper prototypes, and any other artifacts may seem useless to outside observers but will be meaningful to your team. You made these artifacts together and because of that, you have the shared understanding necessary to extract

stories from them. Use them in your IPM (Figure 7-4) to write user stories together, then evaluate and prioritize the stories.

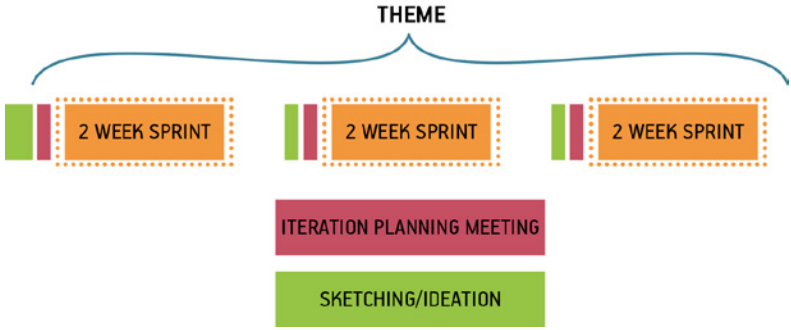


Figure 7-4. *Hold iteration planning meetings immediately after brainstorming sessions.*

User Validation Schedule

Finally, to ensure a constant stream of customer voices to validate against, plan user sessions every single week (Figure 7-5). Your team will never be more than five business days away from customer validation but still your ideas will have ample time to react prior to the end of the sprint. Use the artifacts you created in the ideation sessions as the base material for your user tests. Remember that when the ideas are raw, you are testing for value (i.e., do people want to use my product?). Once you have established a desire for your product, subsequent tests with higher fidelity artifacts will reveal whether your solution is usable.

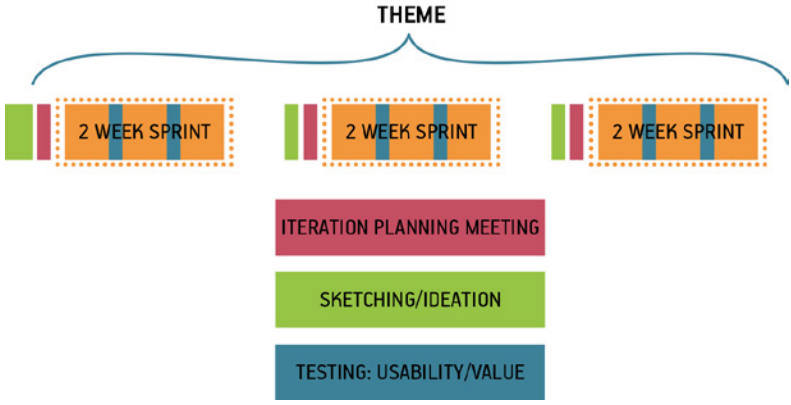


Figure 7-5. *Conversations with users happen every week.*

Participation

One of the big lessons I took away from the diagram I showed at the beginning of Section III was that designers need time to be creative. Two-week cycles of concurrent development and design offer few opportunities for creative time. Some Agile methods take a more flexible approach to time than Scrum does. (For example, Kanban does away with the notion of a two-week batch of work and places the emphasis on single-piece flow.) But you can still make time within a Scrum sprint in which creative activities can take place.

The reason my UX team at TheLadders wasn't finding that time was that we weren't fully participating in the Scrum process. This was not entirely our fault: the content of many Scrum meetings offered little value to UX designers. However, without our participation, our concerns and needs were not taken into account in project plans. As a result, we weren't creating the time within sprints for creative work to take place—the rest of the team didn't understand that this time was needed.

For Lean UX to work in Agile, the entire team must participate in all activities—standups, retrospectives, IPMs, brainstorming sessions—they all require everyone's attendance to be successful. Besides negotiating the complexity of certain features, cross-functional participation allows designers and developers to create effective backlog prioritization.

For example, imagine at the start of a sprint that the first story a team prioritizes has a heavy design component to it. Imagine that the designer wasn't there to voice his or her concern. That team will fail as soon as it meets for its standup the next day. The designer will announce that the story has not been designed. And he or she will say that it will take at least two or three days to complete the design before the story is ready for development. Imagine instead that the designer had participated in the prioritization of the backlog. His or her concern would have been raised at planning time. The team could have selected a story card that needed less design preparation to work on first, which would have bought the designer the time necessary to complete the work.

The other casualty of sparse participation is shared understanding. Teams make decisions in meetings. Those decisions are based on discussions. Even if 90 percent of a meeting is not relevant to your immediate need, the 10 percent that is relevant will save hours of time downstream explaining what happened at the meeting and why certain decisions were made.

Participation allows you to negotiate for the time you need to do your work. This is true for UX designers as much as it is for everyone else on the team.

Design Is a Team Sport: Knowsy Case Study

In this case study, designer and coach Lane Halley details how she brought to the table all the players—development, design, marketing, and stakeholders—to create a tablet game.

In my work as a product designer, I use Lean UX practices on a variety of projects. Recently I've worked on entertainment, ecommerce, and social media products for different platforms, including iPad, iPhone, and Web. The teams have been small, ranging from three to seven people. Most of my projects also share the following characteristics:

- The project is run within an Agile framework (focus on the customer, continuous delivery, team sits together, lightweight documentation, team ownership of decisions, shared rituals like standups, retrospectives, etc.).
- The team contains people with a mix of skills (front- and back-end development, user experience and information architecture, product management and marketing, graphic design, copywriting).
- The people on the team generally performed in their area of expertise/strength but were supportive of other specialties and interested in learning new skills.

Most of the teams I work with create entirely new products or services. They are not working within an existing product framework or structure. In "green fields" projects like these, we are simultaneously trying to discover how this new product or service will be used, how it will behave and how we are going to build it. It's an environment of continual change, and there isn't a lot of time or patience for planning or up-front design.

The Innovation Games Company

The Innovation Games Company (TIGC) produces serious games—online and in-person—for market research. TIGC helps organizations get actionable insights into customer needs and preferences to improve performance through collaborative play. In 2010, I was invited to help TIGC create a new game for the consumer market.

I was part of the team that created Knowsy for iPad, a pass-and-play game that's all about learning what you know about your friends, family, and coworkers, while simultaneously testing how well they know you. The person who knows the other players best wins the game. It is a fast, fun, and truly "social" game for up to six players.

It was our first iPad application, and we had an ambitious deadline: one month to create the game and have it accepted to the App Store. Our small team had a combination of subject-matter expertise and skills in front- and back-end development as well as visual and interaction design. We also asked other people to help us play-test the game at various stages of development.

A Shared Vision Empowers Independent Work

Until a new product is coded, it's hard for people to work within the same product vision. You can recognize a lack of shared vision when the team argues about what features are important or what should be done first. There can also be a general sense that the team is not "moving fast enough," or that the team is going back over the same issues again and again.

While working on Knowsy, I looked for ways that I could make my UX practice more collaborative, visual, lightweight, and iterative. I looked for opportunities to work in real time with other people on the team (such as developers and the product manager) and rough things out as quickly as possible at the lowest responsible level of fidelity.

As we found the right solutions and the team understood and bought into the design concept, I was able to increase fidelity of the design artifacts, confident that we shared a product vision (Figure 7-6).

Figure 7-6. *The Knowsy team with the wall of artifacts behind them.*

Breaking the Design Bottleneck

Early in the project, I sat with the front-end developer to talk about the game design. We created a high-level game flow together on paper, passing the marker back and forth as we talked. This was my opportunity to listen and learn what he was thinking. As we sketched, I was able to point out inconsistencies by asking questions such as, "What do we do when this happens?" This approach had the benefit of changing the dialog from "I'm right and you're wrong" to "How do we solve this problem?"

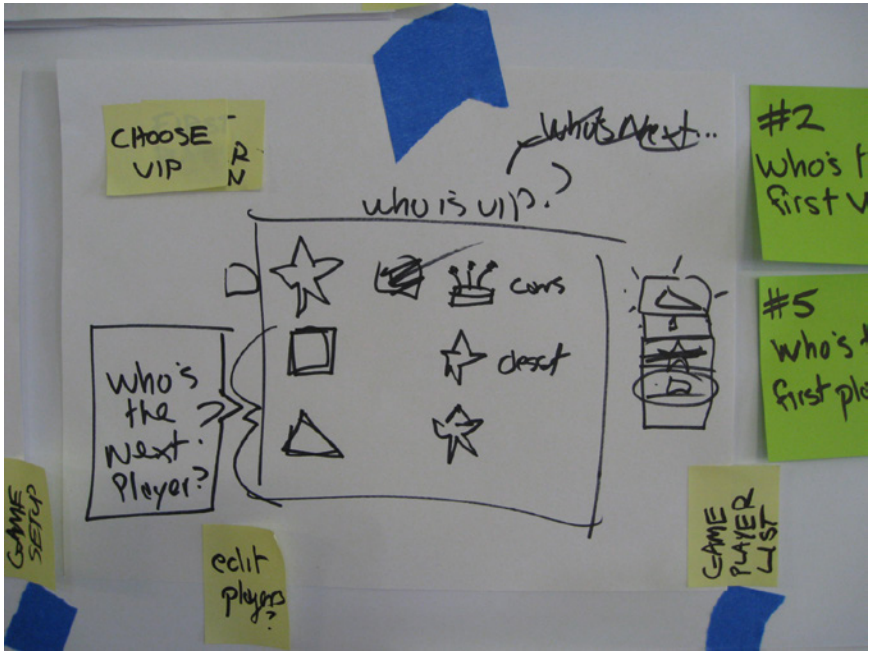


Figure 7-7. *The paper prototype begins to take shape.*

After we had this basic agreement, I was able to create a paper prototype (Figure 7-7) of the game based on the flow and play-test it with the team. The effect on the team was immediate. Suddenly everyone "got it" and was excited about what we were doing. People started to contribute ideas that fit together well, and we were able to identify which parts we could each work on that supported the whole.

Once we were all on the same page, it was easier for me to take some time away from the team and document what we'd agreed in a clickable prototype. See Figure 7-8.

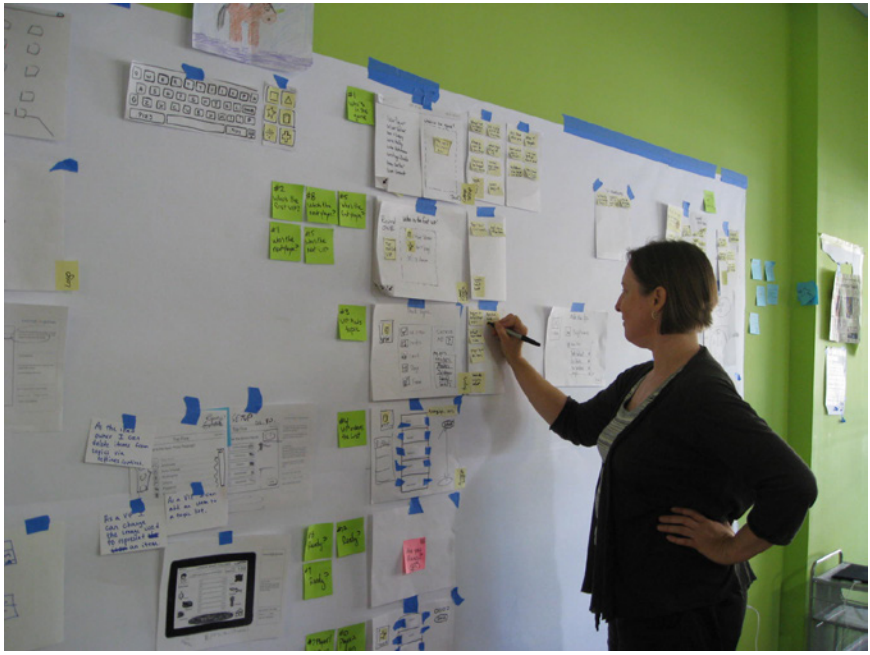


Figure 7-8. *Lane updating the prototype and artifact wall in real time.*

The Outcome

Knowsy's foray into Lean UX proved a success. We got the app to the Apple store by the deadline. I was called back later to help the team do another variant of the product. For that round, I used a similar process. Because I was working remotely and the dev team was not as available to collaborate, I had to make heavier deliverables. Nevertheless, the basic principle of iterating our way to higher fidelity continued.

Beyond the Scrum Team

Management check-ins are one of the biggest obstacles to maintaining team momentum. Designers are used to doing design reviews, but unfortunately, check-ins don't end there. Product owners, stakeholders, CEOs, and clients all want to know how things are going. They all want to bless the project plan going forward. The challenge for outcome-focused teams is that their project plans are dependent on what they are learning. They are responsive, so their typical plan lays out only small batches of work at a time. At most, these teams plan an iteration or two ahead. This perceived "shortsightedness" tends not to satisfy most high-level managers. How then do you keep the check-ins at bay while maintaining the pace of your Lean UX and Scrum processes?

Two words: proactive communication.

I once managed a team that radically altered the workflow for an existing product that had thousands of paying customers. We were so excited by the changes we'd made that we went ahead with the launch without alerting anyone else in the organization. Within an hour of the new product going live, the VP of Customer Service was at my desk, fuming and demanding to know why she wasn't told about this change. Turns out that when customers have problems with the product, they call in for help. Call center reps use scripts to troubleshoot the customers' needs and offer solutions, and they didn't have a script for this new product…because they didn't know it was going to change.

This healthy slice of humble pie served as a valuable lesson. If you want your stakeholders—both those managing you and those dependent on you—to stay out of your way, make sure they are aware of your plans. Here are a few tips:

- Proactively reach out to your product owners and executives.
- Let them know:
  - How the project is going
  - What you tried so far and learned
  - What you'll be trying next
- Keep the conversations focused on outcomes (how you're trending towards your goal), not feature sets.
- Ensure that dependent departments (customer service, marketing, ops, etc.) are aware of upcoming changes that can affect their work.
- Provide them with plenty of time to update their workflows if necessary.

Conclusion

This chapter took a closer look at how Lean UX fits into a Scrum process. In addition, I talked about how cross-functional collaboration allows a team to move forward at a brisk pace and how to handle those pesky stakeholders and managers who always want to know what's going on. I discussed why having everyone participate in all activities is critical and why the staggered sprint model is only a way-point on the path to true agility.

In the next and final chapter, we'll take a look at the organizational shifts that need to be made to support Lean UX. This chapter can serve as a primer for managers on what they'll need to do to set teams up for success.

O'REILLY®

Designing for Performance

WEIGHING AESTHETICS AND SPEED

Lara Callender Hogan

[8]

Changing Culture at Your Organization

The largest hurdle to creating and maintaining stellar site performance is the culture of your organization. No matter the size or type of team, it can be a challenge to educate, incentivize, and empower those around you. Performance more often comes down to a cultural challenge, rather than simply a technical one.

It is rare to have a culture of performance in which everyone at an organization values the impact that performance has on the user experience. Often, there are performance cops or janitors at a company who take it upon themselves to improve site speed. Sometimes, companies will dedicate infrastructure team resources toward performance improvements. There should absolutely be performance champions at your organization (in fact, you're probably one of them!). However, limiting the responsibility of performance to a small group of people will make it nearly impossible to keep the site's speed under control, particularly as the site ages, changes, and is worked on by new people.

It's important to recognize when a problem needs technical solutions, when it needs cultural solutions, and when it needs both. Many of the chapters in this book cover technical solutions for performance, but the cultural solutions covered here will help you leverage these technical solutions' impact and make sure it lasts.

Performance Cops and Janitors

Performance improvements often begin as one person's voice within a company culture. You start to notice how other sites are making optimizations and improving their user experience through tweaks to perceived performance or total page load time. Then you start measuring how your competitors' sites fare in WebPagetest and comparing your

site's performance to theirs. After beginning to learn about many of the easy performance wins that you could implement on your site, you start crafting improvements with little effort and tons of gains.

These are the individuals who often start out as performance cops or janitors. Cleaning up after other designers and developers becomes a routine chore for these individuals; sometimes they've taken this responsibility on themselves, or sometimes they were assigned these responsibilities. Either way, this road leads to burnout.

As time marches on, so many things will continue to create performance challenges for even the most stable site:

- New performance techniques emerge, like the recent implementation of `picture`.
- The site's hardware, brand, and code age.
- New designers and/or developers are hired.
- Existing designers and/or developers with great performance habits leave.
- Browsers continue to evolve.
- Web standards evolve, such as HTTP/2, which eradicates some existing performance constraints.

Having a dedicated team of people responsible for keeping track of these kinds of evolutions is important. A performance champion, or a team of performance champions, is an excellent tool for a company to lean on as the Web changes. But the responsibility for maintaining a high-performing site should not solely rest on the shoulders of these individuals. Everyone who works on the site should buy in to the importance of performance and understand what they can do to improve it.

If other designers and developers who shape the site aren't educated on performance, how can they make the best decisions about user experience? How can they weigh the balance between aesthetics and page speed? If they aren't empowered to make improvements, any performance champions will simply be playing cleanup after other people's work. Spending your time cleaning up other people's work (especially when it's preventable) is a one-way ticket to burnout.

A dedicated performance team can focus on:

- Giving lectures, lunch-and-learns, and workshops to educate others about performance
- Celebrating the good work of designers and developers on other teams who improve site speed
- Building tools to surface performance data in others' daily workflows to help them understand how they are directly impacting performance in their current work
- Defining baseline requirements for performance, such as a performance budget for each new project or a maximum page load time across the site
- Learning about emerging technology and new methods of improving performance
- Communicating publicly about changes in site performance and recent experiments and learnings, as shown in Figure 8-1

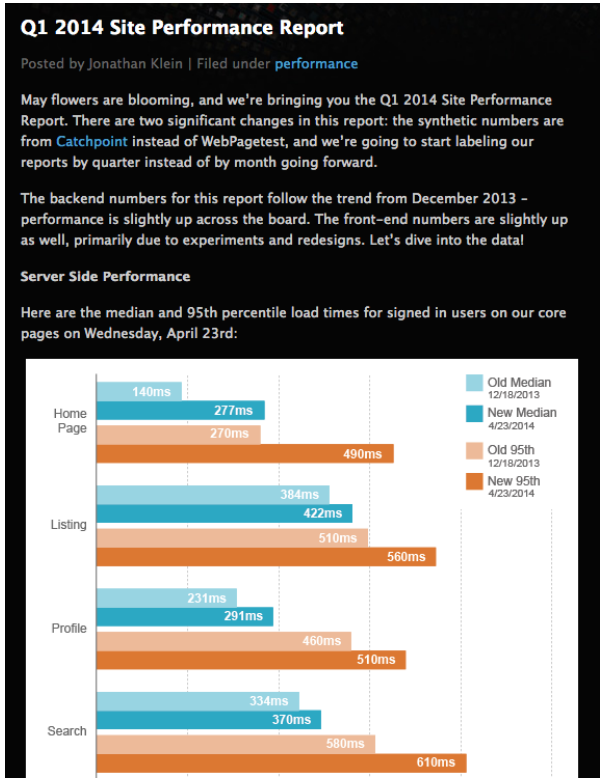


Figure 8-1. Etsy's performance report details load time for top pages and what changes contributed to the load time each quarter.

Having an individual or team care deeply about performance is important for all of the aforementioned purposes. These champions can stay on top of how performance is being handled sitewide; they can keep an eye on problem areas, look for areas to improve, and raise suggestions to the other people contributing to the site's design and development. But the work to be done to actually improve and maintain performance needs to be owned and shared across your organization, rather than lie with an individual or single team.

Upward Management

Page speed is a relatively intangible problem. Though it's easy to get numbers around it, performance is mostly about perception and feeling. Total load time and frames per second don't easily communicate to people *why* they should care about making improvements; problems that are less tangible like this often need a champion within an organization who comes from a place of power. Very Important People who care about performance will help you dramatically shape your organization's culture.

To emphasize the importance of performance upward, focus on showcasing it both within business metrics and with end user experience. The first angle involves numbers: impact on conversion rate, total revenue, returning visitors. The second angle focuses on helping these VIPs *feel* how slow your site is and empathize with your end users.

IMPACT ON BUSINESS METRICS

There are plenty of studies across the Internet that demonstrate the business metric impact of performance, some of which we discussed in Chapter 1:

- Akamai has reported that 75% of online shoppers who experience an issue such as a site freezing, crashing, taking too long to load, or having a convoluted checkout process will not buy from that site (*http://bit.ly/1ttKKNf*).
- Gomez studied online shopper behavior (*http://bit.ly/1ttKspl*) and found that 88% of online consumers are less likely to return to a site after a bad experience. The same study found that "at peak traffic times, more than 75% of online consumers left for a competitor's site rather than suffer delays."

- Users will return to faster sites, as evidenced in a study by Google (*http://bit.ly/1ttKPR8*) that noted a decrease in searches for users who experienced a site slowdown.
- DoubleClick, a Google ad product, removed one client-side redirect (*http://bit.ly/1ttLjqx*) and saw a 12% increase in click-through rate on mobile devices.

Identify what kinds of numbers your upper management cares about. Is it revenue? Membership? Social media engagement? Once you figure out which metrics matter to them, find and share performance research with them that relates to those particular metrics. Correlate engagement metrics (such as bounce rate, click-through rate, and returning visitors) with revenue and other bottom-line metrics that resonate with this audience. Each organization and its VIPs have defined business drivers to which you can draw parallels from these studies.

If possible, run experiments on your own site to correlate performance improvements to the metrics these folks care about and share them alongside the other public research. While big sites like Amazon and Google can run slow-down experiments to measure the impact that a slower site has on its users, your organization probably won't like the idea of you intentionally slowing down the site just to see what happens. Focus on finding high-impact quick performance wins, like compressing images or implementing better caching.

Make one significant improvement and measure its engagement metric impact. If possible, run an A/B test to compare your audience's behavior in the control to your new, improved variant. If you're able to move the needle on revenue-related metrics like conversion rate, terrific; if not, focus on other engagement metrics like bounces and pages per visit. Tie any statistically significant improvements in your new high-performing version to the metrics that upper management cares about. A lower exit rate, for example, could mean more users choosing you over a competitor or returning to search engine results.

If you're unable to run an A/B test, measure engagement metrics before you make the improvement and again afterward. It won't be scientific, but it'll be the best case you can make to upper management. Read more about measuring the impact of performance improvements in Chapter 6. Share the work you did and the resulting business metric changes with those VIPs to help them understand the impact that performance work can have.

As you make any performance changes, also measure how long it takes you to do so. Design and development hours are a cost for the business, and you'll need to address this as you work on turning VIPs into champions of your cause. Find the quickest and most impactful wins possible to start to emphasize that improving the user experience doesn't have to be a large cost to the business. Translating a specific number of resource and development hours into a revenue win for the business will be your biggest asset in the conversation, and will help you continue to get support as larger and more time-intensive performance work is needed.

Conversations with upper management should include a blend of public research from around the Internet, research that you've done on your own site, as well as the cost of this kind of work to the business. A holistic approach to these conversations should be grounded in an understanding of which engagement metrics and business factors resonate the most with your internal audience.

EXPERIENCING SITE SPEED

Helping upper management understand what your users are experiencing on your site is key. We can talk numbers all day, but getting to the root of how your performance affects your users will require you to focus on your site's user experience. Remember that most people within your organization are probably accessing your site on newer hardware with fast connections and are probably relatively close to your datacenter. How do people around the globe experience your site? How do people experience your site when they're not on a desktop computer?

Run multiple WebPagetests using different locations and devices and compare the results. You can compile all of the results into a single filmstrip view to compare them using this URL structure: *webpagetest.org/video/compare.php?tests=<Test 1 ID>,<Test 2 ID>...*

For example, in Figure 8-2 I ran three separate tests for the *Huffington Post*'s site: one using the Virginia test location using Chrome on a desktop, one using Internet Explorer 8 from the Singapore location, and one using Chrome on an Android phone from the Virginia location. While the overall numbers varied widely for each test and could make a compelling argument for mobile and global performance improvement needs, the filmstrip view really helps you *feel* the difference in user experience.

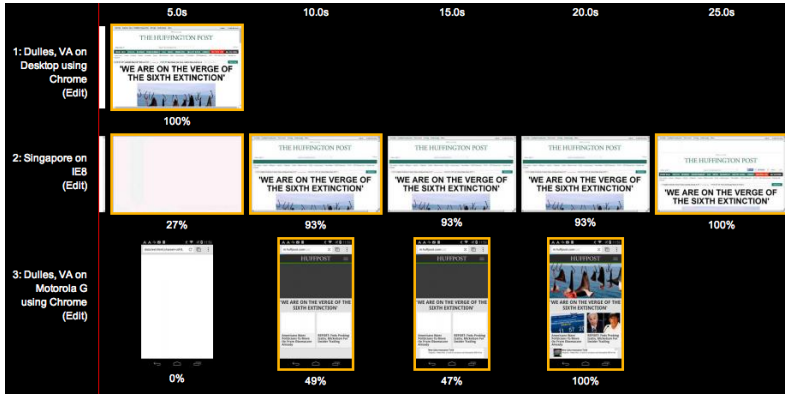


Figure 8-2. WebPagetest provides a filmstrip view as well as video for you to compare tests at the same time. This helps give you a better understanding and feel for the performance of these sites.

Another angle to consider during these conversations is pride. While revenue impact is a great metric you can use to convince upper management that performance should be an important consideration for any designer and developer at your organization, it's not the only tool in your tool belt. Your site likely has competitors. How do their page load times compare?

WebPagetest also allows you to compare multiple URLs before you begin a test for a visual comparison of performance (see Figure 8-3). All of these tests in the Visual Comparison tool will use the Dulles, Virginia, testing location.

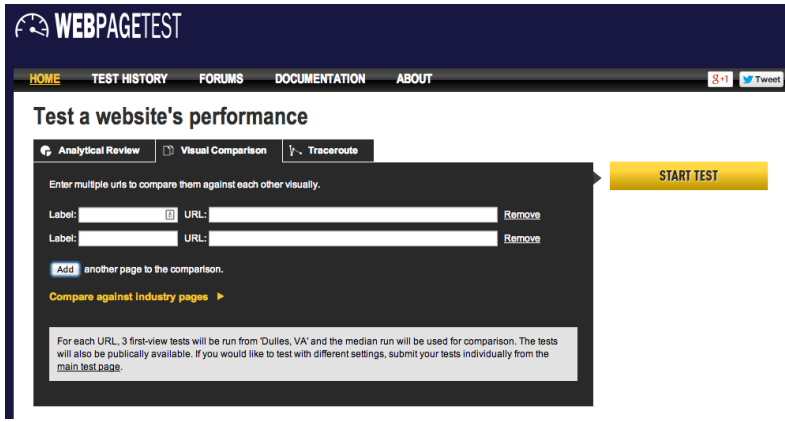


Figure 8-3. You can enter multiple URLs into WebPagetest to compare their performance.

Once the tests complete, WebPagetest can show you a filmstrip view of how each page loads over time, as shown in Figure 8-4. You can even export a video of the page loads in tandem; this really helps people *feel* the difference of how the sites are loading. You're able to avoid numbers altogether, instead focusing on gaining an understanding of how your users are *experiencing* your site and your competitor's site in the same time frame.

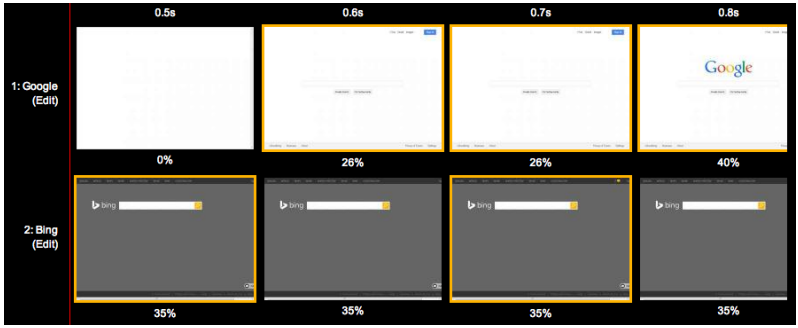


Figure 8-4. Use WebPagetest's filmstrip view and video comparison to gain a better understanding and compare the performance feel of different sites.

Page speed and user experience are not secrets. Any of your competitors can test your site, or run it through performance tools and see how you're stacking up. Remind the Very Important People in your organization that you are being analyzed not just by your users, but by your competitors, too. Be sure that you are outperforming your competitors' sites.

One last way to utilize the filmstrip and video views is to compare the before and after of a performance improvement you make to your site. While measuring the impact that the improvement had on engagement metrics is powerful, it can be equally helpful to document the visual of how differently the site loads, particularly if your improvement helps perceived performance rather than total page load time.

Use these tools during your conversations with upper management to help make the case that everyone at your organization has an impact on the end user experience and should focus on performance as part of their daily work. A site that feels fast requires everyone who affects the user experience to keep performance top-of-mind during their daily workflows.

Working with Other Designers and Developers

Education and empowerment are key to incentivizing the other designers and developers with whom you work to care about performance. The responsibility is on you to continually equip them with the tools that they need as well as the reasons to care about how they impact user experience when they affect your site's performance. While it's true that hammering home the negative consequences of poor performance will help make it clear how important it is, championing and celebrating performance wins is often way more successful in the long run. Help those around you care about delivering an awesome user experience and know how valuable their work is as it impacts performance.

EDUCATING

There are many ways that focusing on performance helps designers and developers. Considering things like semantics and repurposability of what's being built up front saves a ton of design and development time later. The ease of editability increases, and future headaches are prevented when code is cleaner and design patterns can be easily updated across the site at once or repurposed.

Beyond these wins, you'll need to educate others at your organization about how they impact performance in their daily lives. Brown-bag lunch sessions, lectures, and workshops are all excellent ways to communicate to and train people about how they can be better designers and developers by focusing on performance. Consider leading an effort to teach people about topics such as:

- How mobile performance works
- How people can impact performance during the design stage
- How to improve perceived performance

Share slide decks and presentation videos from others about how to design excellent, high-performing user experiences. Education is an ongoing effort; you'll have new hires who are unfamiliar with these techniques, and folks who forget about best practices when they get swamped with other work. Routinely give lunch-and-learns or other informal education about how everyone can have a positive impact on performance.

Develop baselines for your organization as to what's acceptable for page load time. How slow is too slow? Communicate the acceptable page load time threshold to everyone: "We're aiming for one-second total page load time for each page." Alternatively, assess what the best-performing pages on your site are and how fast they load, and use that as a benchmark across the site. Be sure to measure the worst-performing pages that get a lot of traffic on your site and suggest that the entire team focus on getting those as fast as possible. People should be given easy-to-follow guidelines and benchmarks so that it's clear where the wins are and what to aim for.

If you're able to run automated tests to gather performance information for your most important pages, do so. Make sure the team has visibility into when a page's performance gets worse so that you can figure out what changes contributed to the decline and fix them. Set up alerts on worsening performance and share them with other designers and developers so that everyone can learn as the site evolves.

For each new project, develop a performance budget and make sure all designers and developers understand what it means. Educate them about these numbers and how they can weigh aesthetics and speed. Read more about performance budgets in "Approach New Designs with a Performance Budget." Providing baseline guidance and easy-to-understand (and easy-to-measure) metrics for the entire team will empower them to contribute to a stellar user experience.

EMPOWERING

To empower people to make good choices during their daily workflows, figure out how to surface performance data on their current work. At Etsy, we have a toolbar that appears when an Etsy employee is logged in to the site, as shown in Figure 8-5. Designers and developers use this toolbar to understand information about the page they're looking at as they work on the page; it includes visit traffic data, a list of any experiments that are currently being run on the page, and tools to view the mobile version of the page. It also includes performance timing data and an alert whenever the performance times violate our performance service-level agreements.

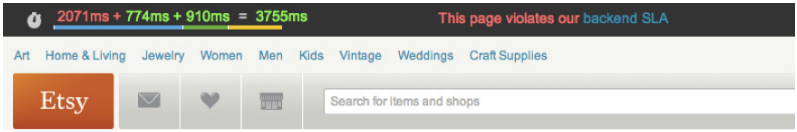


Figure 8-5. At Etsy, we show a toolbar to employees working on site pages. It surfaces performance timing data and makes it clear when a page has performance problems so that the designer or developer working on the page is alerted to the issue.

Showing performance data in this way is helpful to designers and developers, as it is a constant reminder that performance is part of the user experience. Rather than waiting to see how fast a site is after it's been built, consider ways to routinely empower designers and developers with this knowledge as they're working.

Another way to routinely share this information is to send automated emails if any performance regressions occur across the site. Equipping people with this knowledge *as it happens* is an important step toward empowering people to immediately fix it. Make this kind of performance metric knowledge a part of daily life and workflows so that it feels natural, like it's just part of doing a good job at work.

Once people have the tools and education needed to understand the performance of your site and how they can impact it, they'll begin to feel empowered to improve it. But remember, this is a cultural problem, not a technical one; though there are a lot of technical solutions that can help people improve site speed, you'll need to do extra work to solve the social aspects of performance culture.

One way to change the culture at your organization is to begin to publicize your performance efforts. When I worked at Dyn, I published a summary of how I completed a huge template cleanup and included the performance improvements that resulted. It not only helped educate the readers of Dyn's blog, but it also made the performance win highly visible to all Dyn employees.

When frontend architect and consultant Harry Roberts completed a chunk of performance work for a client, he shared the numbers with them. "They got very, very excited about the numbers, and even began running their own tests on it. Giving them something like this to get into really brought them on board so, from then on in, they cared as much about keeping the numbers down as I did," said Roberts.

Publishing your work and celebrating it is a huge incentive to many designers and developers; showcasing improvements, as I did in Figure 8-6, is a great way to kick-start culture change and encourage others to contribute to performance wins.

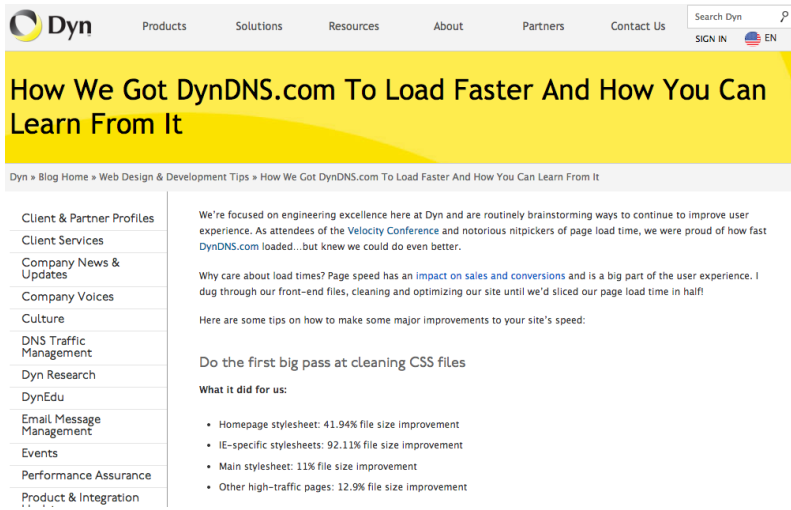


Figure 8-6. After I finished a template cleanup effort across DynDNS.com, I published a summary of how I did it and the performance improvements we saw.

At Etsy, the performance team attempted a different public tactic to effect culture change. In 2011, the team published its first performance report that included an outline of load times for the top pages on the site, which you can see in Figure 8-7. It included some relatively embarrassing metrics, but the performance team realized that it was important to acknowledge the opportunities for performance improvement. They recognized that site speed is not a secret—it can be measured by anyone—and these numbers were important for everyone at Etsy to recognize because they reflected the site's actual user experience.

After publishing the first report, the team responsible for working on the home page realized how embarrassing their numbers were. They worked on improving load time by making some hard decisions about features and how they were designed, weighing the balance between aesthetics and speed. They were able to reduce the home page's load time significantly by the time the next performance report was released, as shown in Figure 8-8.

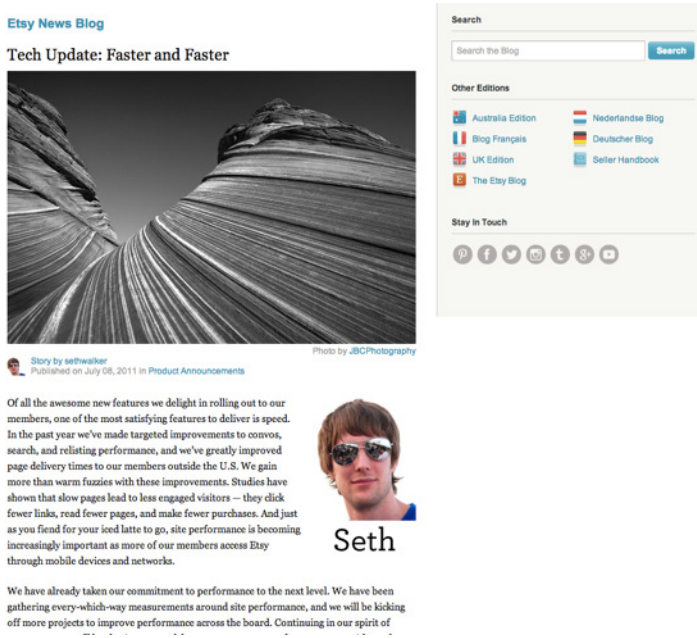


Figure 8-7. Etsy published its first performance report in 2011, intentionally including some embarrassingly long page load times.

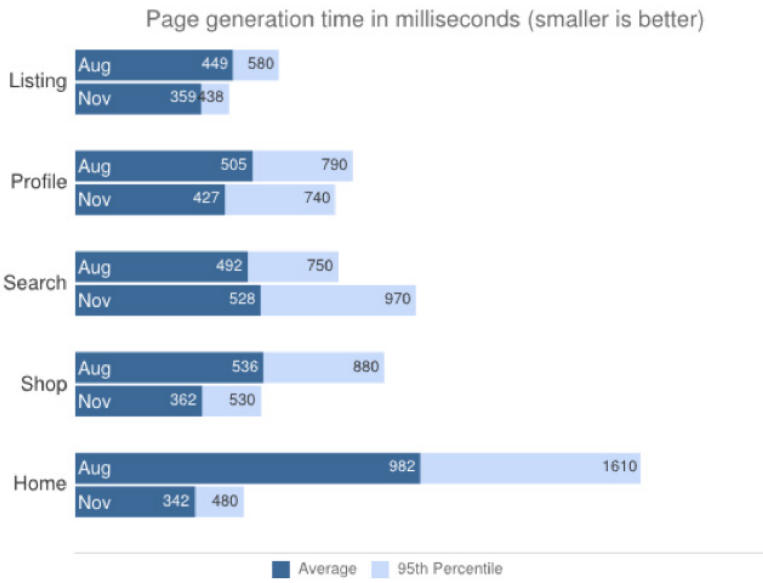


Figure 8-8. In its second performance report, Etsy showcased huge improvements in home page load time.

Publicly acknowledging how your site is performing will make people feel accountable, and will also make them want to help. Designers and developers generally want to help contribute to a common, positive cause, and making this cause public will help kick-start this feeling.

Another way to help kick-start the culture shift is to make it very easy for the team to feel productive when making performance improvements. Find all of the low-hanging fruit across the site—that is, work that could be easily picked up by another designer or developer—and start documenting it. File tickets or start a list that people can quickly reference. Here are some examples of easy performance wins you can share:

- Clean up and normalize existing button styles across the site, and document where all the different buttons live so people can pick them off one by one.
- Isolate suspect chunks of CSS that are likely no longer needed in your stylesheet and ask someone to verify that they're no longer needed, then have folks clean them out.
- Find large images used on the site and list them so that someone can re-export them, compress them, or find other ways to optimize their file size.

For each ticket or item on your list, include enough detail about the fix needed so that someone picking it up can immediately work on the solution. Keep each piece of work bite-sized, no more than a few hours each. If a fix takes more than a few hours, ask the designer or developer to simply document the progress so that another person can pick it up again in the future. It should be intuitive and easy for other designers and developers to begin contributing to making your site faster.

As others begin contributing to the overall performance of your site, the most important thing you can do is celebrate their work. For every bite-sized performance improvement, thank the contributor and publicize their work internally, like in Figure 8-9.

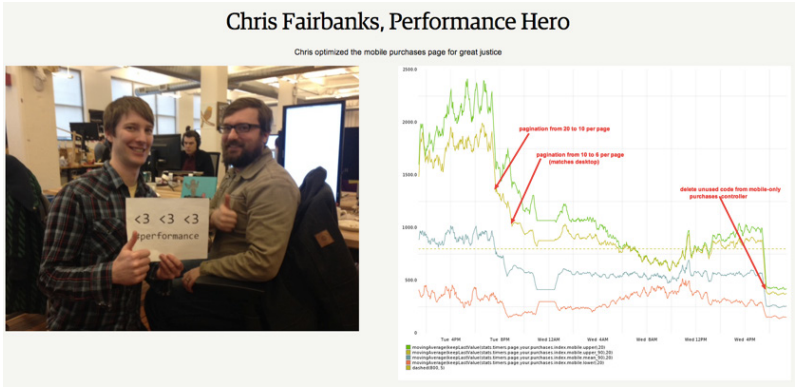


Figure 8-9. The performance team at Etsy maintains a dashboard celebrating people on other teams who contribute to performance improvements. We include their photo, a graph showing the performance improvement, and a brief description of their solution.

At Etsy, we maintain an internal dashboard where we can celebrate "performance heroes": people on other teams who contribute fixes and improvements to our site's page load time and perceived performance. We routinely update it to showcase the creative efforts of the people with whom we work, highlighting any relevant graphs that illustrate the performance improvement and a description of the solution they implemented. We'll also send out an email to the other designers and developers at Etsy to indicate we've updated the dashboard so that everyone can chime in and high-five the person who improved the site.

Performance is truly everyone's responsibility. Anyone who affects the user experience of a site has a relationship to how it performs. While it's possible for you to single-handedly build and maintain an incredibly fast experience, you'd be constantly fighting an uphill battle when other contributors touch the site and make changes, or as the Web continues to evolve. Educate and empower everyone around you to understand how they can improve performance, and how their choices affect the end user experience. Performance truly is about making a cultural shift, not just a technological one; build performance champions within your organization so that you can create the best user experience possible for your site.

Web performance work is as fulfilling as it is challenging. You have the power to go and create an excellent experience for your users. Find those performance wins, whether they're implementing new caching rules, optimizing images, or creating repurposable design patterns. Empower those with whom you work to be performance champions. Strive for the best possible user experience, striking a balance between aesthetics and speed. With a focus on performance, everyone wins.

O'REILLY®

Creating a Data-Driven Organization

PRACTICAL ADVICE FROM THE TRENCHES

Carl Anderson

CHAPTER 1
What Do We Mean by Data-Driven?

> Without data you're just another person with an opinion—William Edwards Deming

> It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts—Sherlock Holmes

In this book, I address two core questions:

- What does it mean for an organization to be data-driven?
- How does an organization get there?

Many organizations think that simply because they generate a lot of reports or have many dashboards, they are data-driven. While those activities are part of what a data-driven organization does, fundamentally they are descriptive and backwards-looking. They state what happened but they are not prescriptive. As such, they have limited upside. Having reports and dashboards is nothing without considering what happens to the information they contain. If they are unread, ignored, and the boss is going to do whatever he or she wants to do, regardless of what the data say, then they are not driving anything. In contrast, consider more forward looking *analysis*, such as predictive models that optimize ad spend, supply chain replenishment, or minimize customer churn. Those insights and recommendations, if acted upon, have a huge potential upside and impact to the organization. However, this is predicated that they are the right data to collect, that the data are trustworthy, the analysis is good, that the insights are considered in the decision, and they drive concrete actions so the potential can be realized. Phew! I call this sequence the "analytics value chain." To be data-

driven, data and insights have to flow down the whole chain, all the way to actions that drives outcomes and impact.

While many organizations may claim to be data-driven, the reality is that relatively few genuinely are and that we could all do better. In this book, I'll discuss the hallmarks of great data-driven organizations. I'll cover the infrastructure, skills, and culture needed to create organizations that take data, treat it as a core asset, and use it to drive and inform critical business decisions and ultimately make an impact. I will also cover some common anti-patterns, behavior that inhibits a business from making the most from its data.

The first step in becoming data-driven is to realize that reporting and alerts, which form the basis of many organizations' data activity, are not in fact data-driven; they are typically a declaration of past or present facts without a great deal of context, without causal explanation of *why* something has or not happened, and without recommendations of what to do next.

Analytics, on the other hand, which involves answering the why questions—or more generally, w-questions: who, what, when, why, and where—making recommendations and predictions, and telling a story around the findings, is a key driver in a data-driven organization. However, even analytics is not necessarily data-driven if its findings are never seriously considered or acted upon. To be data-driven, an organization must have the right processes and the right culture in place to augment or drive critical business decisions with these analyses and so have a direct impact on the business.

Culture, then, is the key. This is a multi-faceted problem that involves data quality and sharing, analyst hiring and training, communication, analytical organizational structure, metric design, A/B testing, decision-making processes and more. This book will elucidate these ideas by providing insights and illustrative examples from a variety of industries. I also bring in the voices of experience in interviews that provide advice and insights of what worked and what didn't from a variety of data science and analytics leaders. I hope to inspire all our readers to become more data-driven.

Moreover, throughout the book I emphasize the role that data engineers, analysts, and managers of analysts can play. I suggest that a data-driven organization and requisite culture can and should be built not only from top-down leadership but also from the bottom up. As Todd Holloway, head of data science at Trulia remarked at the 2014 CDO Executive Forum, "The best ideas come from the guys closest to the data." Not only are they the ones who work directly with the data sources and who recognize and can remedy the data quality issues and understand how best to augment the data, but "they often come up with the good product ideas." In addition, they can help educate the rest of the organization to be more data literate. Part of that comes from developing their skill set and using it to do good work. Part, however, comes from being more business savvy, learning the right questions to ask and business problems to tackle and then "selling" their insights and recommendations to the

decision-makers and making a compelling case of what the finding or recommendation means to business and why it makes an impact.

And there are great impacts and gains to be had. One report[1], controlling for other factors, found that data-driven organizations have a 5–6% greater output and productivity that their less data-driven counterparts. They also had higher asset utilization, return on equity, and market value. Another report[2] claims that analytics pays back $13.01 for every dollar spent. Being data-driven pays!

Data-drivenness is not a binary but rather a continuum: you can always be more data-driven, collect more high quality relevant data, have a more skilled analytics organization, and do more testing. Moreover, you can always have a better decision-making process.

Goals of the book, then, are to inspire the analyst organization to play their part, to provide pause for thought—to ask yourself "are we making the most of our data?" and "can we be more data-driven?"—and to stimulate discussion in your organizations about what more can be done to make use of this key resource. It is never too early to be thinking about this. Senior management and founders should be working to bake this into the very fabric of their company at an early stage. So, let's find out more about what is entailed.

Who should read this book?

The information here will help you build and run an internal analytics program, deciding what data to gather and store, how to access it and make sense of it, and, most crucially, how to act on it.

Whether you're the only data scientist at a startup (and wearing a half-dozen other hats, to boot!), or the manager at an established organization with a room—or a department—full of people reporting to you, if you have data and the desire to act more quickly, efficiently, and wisely, this book will help you develop not just a data program but a data-driven culture.

Early Release Status and Feedback

This is an early release copy of *Creating a Data-Driven Organization*. The text, figures, and examples are a work in progress, and several chapters are yet to be written. We

1 Brynjolfsson, E., Hitt, L. M., and Kim, H, H. 2011. Strength in numbers: how does data-driven decisonmaking affect firm performance? Social Scienc Research Network.http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1819486.

2 Nucleus Research. 2014. Analaytics pays back $13.01 for every dollar spent. http://nucleusresearch.com/research/single/analytics-pays-back-13-01-for-every-dollar-spent/.

are releasing the book before it is finished because we hope that it is already useful in its current form, and because we would love your feedback in order to create the best possible finished product.

If you find any errors or glaring omissions, if you find anything confusing, or if you have any ideas for improving the book, please email the author and editors at *feedback@datadrivenorg.net*.

The Big Picture

Data-drivenness is about building tools, abilities, and, most crucially, a *culture* that acts on data. This chapter will outline what that sets this type of organization apart. I start with some initial prerequisites about data collection and access. I then contrast reporting and alerting versus analysis in some detail because it is such an important distinction. There are many different types of forward looking analysis, varying in degrees of sophistication. Thus, I spend some time going over those types describing them in terms of "levels of analytics" and "analytics maturity," in particular discussing the hallmarks of an analytically mature organization. What does that look like?

Let us start us on the way to answering the first question: what does it mean for an organization to be data-driven?

Data Collection

Let's get a couple of obvious prerequisites out of the way.

Prerequisite #1: An organization can't be data-driven if it is not collecting data.

Data undoubtedly are a key ingredient. Of course, they can't just be any data they have to be the *right* data. The dataset has to be relevant to the question at hand. It also has to be timely, accurate, clean, unbiased, and perhaps most importantly, it has to be trustworthy.

This is a tall order. Data are always more dirty than you imagine. There can be subtle hidden biases that can sway your conclusions, and cleaning and massaging data can be a tough, time-consuming, and expensive operation. I often hear that data scientists spend 80% of their time obtaining, cleaning, and preparing data, and only 20% of their time building models, analyzing, visualizing, and drawing conclusions from those data (for example, http://goo.gl/agby8V and http://goo.gl/urYVE). In my experience, this is entirely plausible. In the next chapter, I'll cover aspects of data quality in much more detail.

Even if you do have quality data, and even if you have *a lot* of quality data, that will only get you so far and, despite the hype that you might hear, does not make you data-driven. Some people, especially certain big data vendors and service providers,

pimp big data as a panacea: if you collect everything, somewhere in there are diamonds (or golden nuggets or needles or one of many other metaphors) that will make any company successful. The hard truth is that data alone are not enough. A small amount of clean, trustworthy data can be far more valuable than petabytes of junk.

Data Access

Prerequisite #2: an organization can't be data-driven if the data aren't accessible and queryable.

Having accurate, timely, and relevant data though is not sufficient to count as data-driven.

- **Joinable**: The data must be in a form that can be joined to other enterprise data when necessary. There are many options such as relational databases, NoSQL stores, or hadoop. Use the right tool for the job. For instance, for a long while, the financial analysts at Warby Parker were using Excel to compute the key metrics reported to senior management. They sucked down huge amounts of raw data from different sources and set VLOOKUPS (an Excel function to find cross-references in the data) running to join all that data to get a top-level look at the numbers. This worked well initially, but as the company's sales and customer base were scaling rapidly, the data got larger and larger, the Excel file approached 300MB, their computers maxed out their RAM, and the VLOOKUPS would take ten hours or more, frequently crash, and had to be restarted. They had stretched the tool and approach as far as it could go. It had been an appropriate tool but company hypergrowth changed that. The mechanics of getting those numbers became a huge time-sink for the analysts and a source of stress as to whether they would get their numbers or have to wait another ten hours to rerun those VLOOKUPS. It turned them from analysts into Microsoft data engineers. My team helped to bring that whole dataset into a MySQL relational database. We wrote queries to crunch the numbers for them, allowing them to focus on analysis, trends, and presentation of those data—a far better use of their time. Now that they have better tools and more time they are producing deeper, richer analyses.
- **Shared**: There must be a data sharing culture within the organization so that data can be joined, such as combining a customer's clickstream to their transactional history. Imagine a patient admitted to a hospital E.R., receiving treatment and then being released with a requirement to attend an outpatient clinic for additional treatment and checkups. The patient is going to receive worse customer service and more importantly worse care if the hospital and clinic don't share data—when, where, and why was he admitted, what issues did he present, what treatment did he receive etc. From the healthcare providers' perspective, their analysts are going to find it hard or impossible to analyze and improve the pro-

cess and care if they don't have a coherent, accurate picture of patient flow, diagnostic processes, and complete longitudinal data of those patients. So, siloed data are always going to inhibit the scope of what can be achieved. This is a case where the whole is greater than the sum of the parts.

- **Queryable**: There must be appropriate tools to query and slice and dice those data. All reporting and analysis requires filtering, grouping, and aggregating data to reduce the large amounts of raw data into a smaller set of higher-level numbers that help our brains comprehend what is happening in a business. I need to be able to see trends or understand differences among customer segments. Analysts have to have tools that allow them to compute those metrics relatively easily.

(All of these topics will be covered in greater detail in later chapters.)

OK, so now we have data and it is accessible. Is this sufficient? No, not yet. You need people with the right skills to use those data. That can mean the mechanics of filtering and aggregating data, such as through a query language or Excel macros, but it also means people who design and choose the appropriate metrics to extract and track (this topic is covered in ???). Those metrics might be resubscription rates (for subscription services such as Netflix or the Wall Street Journal), lifetime values, or growth metrics, but someone needs to decide upon them and someone (else) needs to create some process to provide those values.

So, for an organization to be data-driven, there have to be humans in the loop, humans who *ask the right questions* of the data, humans who have the skills to extract the right data and metrics, and humans who use those data to inform next steps. In short, data alone are not going to save your organization.

Reporting

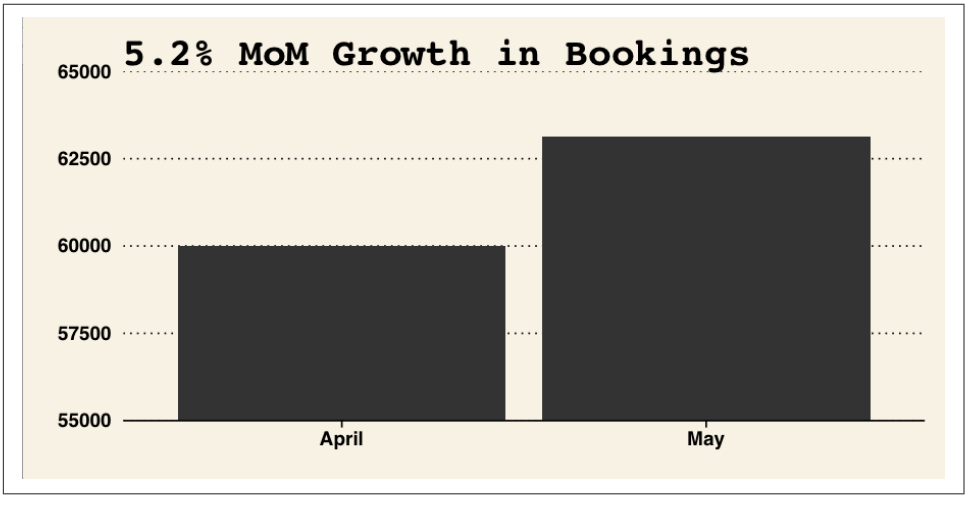


Figure 1-1. 5.2% month over month sales growth! (See text).

Let's suppose you have an analytical group with access to accurate data, they extract sales data and generate a report proudly claiming that the company's bookings grew 5.2% from April to May (Figure 1-1). This is certainly now sounding more like a data-driven company. However, this is still deeply insufficient. Certainly it is good that they are tracking these metrics. The CFO and CEO will definitely be interested in those numbers. What, however, does this value of 5.2% really tell you? Very little, in fact. There are many possible reasons why the company sales grew by this amount:

- Suppose that you sell a highly seasonal product such as beachwear. Maybe 5.2% is much lower than normal. Maybe most years, May's growth is more than 7% over the prior month and this year's growth is well below average.
- Maybe your chief marketing officer spent a lot of money on a national campaign to drive brand awareness. How much of that 5.2% growth was generated from that campaign and was that campaign a good value for the money?
- Maybe your CEO appeared on *Good Morning America* or your product was featured in *Techcrunch* or a video went viral and that was the driver. That is, growth can be traced back to a specific unusual driving event (which might drive transient or sustained growth).
- Maybe monthly sales are low in volume and highly variable. Maybe that growth was just luck and maybe the overall trend is *downwards*. (If you have ever traded stocks, you will surely understand.)

All of these are possibilities, potential reasons as to why something happened. The reported number is just that, a numerical value with little to no context.

> "As orgs grow larger and complex, people at the top depend less on firsthand experience, and more on heavily processed data." —John Gardner
>
> — John Maeda (@johnmaeda) August 16, 2014

Alerting

Ding, ding, ding! Web-app server #14 has just averaged more than 98% CPU utilization for the last 5 minutes.

Alerts are essentially reports about what is happening right now. They typically provide very specific data with well-designed metrics but like reports, they don't tell you why you are seeing a spike in CPU utilization and they don't tell you what to do, right now, to rectify the problem. As such, like reports, they lack this crucial context. There is no causal explanation. This is the point at which performance engineers or system administrators dive into the production logs to ascertain what is going on, why it is happening, and what are the options to fixing it: rollback some code, spin up some more servers, reconfigure the load balancer etc.

Figure 1-2 shows an example of server load over time. There is some variability but most of the day is spent with a run queue of about 0.5 or fewer. At 1am, load starts to spike, shooting up to more than 5, a 10× increase over "normal," in the space of 30 minutes. It seems highly unusual. What's going on? Maybe someone should fix that, but how?

In this case, it is just the weekly backups running. This happens every Thursday at 1am, perfectly normal, nothing to see here. This makes the point that there is great data here, a good metric presented clearly but the context—that it is caused by backups, that it happens on a particular schedule and this 1am time slot is expected, and that the server can handle this load without problems—is all lacking.

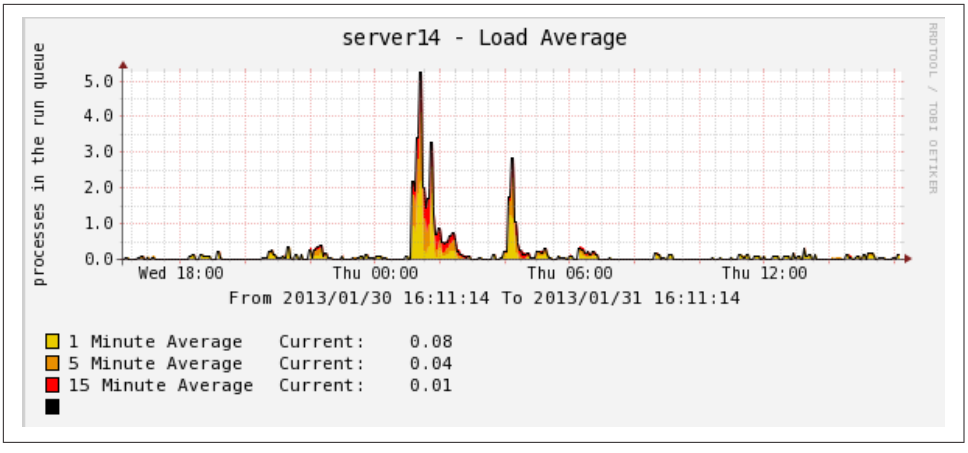


Figure 1-2. Example of server load over time (from http://www.bwfhosting.com/we-got-your-back/). You can see that the load shot up at about 1am, 10× "normal" looking peaks. What is the cause and what should be done about it? An alert doesn't tell you. In this case, this was just the backups running and is perfectly normal behavior.

From Reporting and Alerting to Analysis

Reporting and alerting are necessary but not sufficient characteristics of being data-driven. We should, however, not understate the importance of both these activities. Reporting especially is a highly valuable component of a data-driven organization. You can't have an effective data-driven organization without it. However, the reverse is not true: there are many organizations that focus on reporting and may have little to no real (objective) analysis. For one thing, reporting may be driven by legal requirement and responsibilities, such as Sarbanes-Oxley compliance and generating earnings reports to shareholders, and not from an internal cultural drive to improve the business.

Reporting tells you what happened in the past. It also provides a baseline from which to observe changes and trends. It can be interesting and can keep some investors and shareholders happy but it is a fundamentally backwards view of the world. To be data-driven you have to go beyond that, to be *forward-looking* and engage in analysis, dig in, and find out why the numbers are changing and, where appropriate, to make testable predictions or to run experiments to gather more data that will shed light on why.

Let's be more explicit and compare and contrast the two. Here is one set of definitions:

- **Reporting:** "The process of organizing data into informational summaries in order to monitor how different areas of a business are performing" [3].
- **Analysis:** "Transforming data assets into competitive insights, that will drive business decisions and actions using people, processes and technologies" [4].

Reporting says what happened—we hit a peak of 63,000 simultaneous visitors on the website on Thursday at 10:03am.

Analysis says why did it happen—the company was mentioned in a piece on the TV newsmagazine show "60 Minutes" at 10:01am—and recommends what the organization can or should do to generate more / less of the same.

Reporting is matter of fact, descriptive. Analysis, on the other hand is prescriptive.

Table 1-1. Key attributes of reporting versus analysis. Mostly gleaned from Dykes, 2010

Reporting	Analysis
Descriptive	Prescriptive
What?	Why?
Backwards-looking	Forwards-looking
Raise Questions	Answer questions
Data → Information	Data + information → insights
Reports, dashboards, alerts	Findings, recommendations
No context	Context + storytelling

In Table 1-1, we summarize the differences between the two. Hopefully it is now clear why analysis and being data-driven is such a powerful facet or cultural component of a business. This is what can drive a business in new directions or to greater levels of efficiency.

3 Dykes, B. 2010. Reporting vs. Analysis: What's the Difference? http://blogs.adobe.com/digitalmarketing/analytics/reporting-vs-analysis-whats-the-difference/.

4 Faria, M. 2013. Acting on Analytics: How to Build a Data-Driven Enterprise. https://www.brighttalk.com/webcast/1829/80223.

Table 1-2. Davenport's hypothesized key questions addressed by analytics (modified from Davenport et al., 2010). D) is valuable analytics but only E) and F) are data-driven and then if and only if information is acted upon (more explanation in text).

	Past	Present	Future
Information	**A) What happened?** reporting	**B) What is happening now?** alerts	**C) What will happen?** Extrapolation
Insight	**D) How and why did it happen?** Modeling, experimental design	**E) What's the next best action?** Recommendation	**F) What's the best/worst that can happen?** Prediction, optimization, simulation

A useful framework for understanding analytics is Davenport et al. ([5]; see Table 1-2). Here we can see insight-driving activities in the bottom row. As I noted above, reporting (A) and alerting (B) are simply not data-driven, they state what happened in the past or that something unusual or undesirable is happening now; there is no explanation of why it is happening or why it did happen, and no recommendations as how to resolve or reproduce the situation. Digging down to understand causal factors through models or experiments (D) is a precursor to data-drivenness. Only by understanding why something happened can you formulate a plan or set of recommendations (E). E) and F) are truly data-driven but if and only if the information is acted upon—explained in detail below.

(C is a danger zone: it can be easy enough to extend a perceived trend out to the future—in Excel, click "Chart" and then "Add trendline"— that is, extrapolate outside the current's data range and make a naïve prediction. Even making a sensible choice about a functional form for the model, there are many reasons why that prediction may be misleading or plain wrong. To have confidence in those predictions you should strive to have a causal model. Types of analysis is covered in ???.)

In summary, the bottom row highlights forward-looking activities that include elements of causal explanation. Now we are starting to touch upon what it means to be data-driven.

Hallmarks of data-drivenness

There are several types of activities that truly data-driven organizations engage in:

5 Davenport, T. H., Harris, J. G., and Morison, R. 2010. *Competing on Analytics*. Harvard Business Press, Boston.

- A data-driven organization may be continuously testing. It might be A/B testing checkout flow on a website or testing email subject lines in a marketing campaign. LinkedIn, for instance, runs 200 experiments per day while Etsy run dozens of experiments simultaneously. Tests may also include user testing, working directly with actual customers or users to obtain direct feedback on possible new features or products.
- A data-driven organization may have a "kaizen" or continuous improvement mindset. It may be involved in repeated *optimization* of core processes, such as shaving minutes off manufacturing times or decreasing cost per acquisition. This comes about through careful analysis, crafting mathematical or statistical models, and simulation.
- A data-driven organization may be involved in predictive modeling, forecasting sales, stock prices, or company revenue but importantly feeding the prediction errors and other learning back into the models to help improve them (we cover this further in ???).
- A data-driven organization will almost certainly be choosing among future options or actions using a suite of weighted variables. Resources are always finite and there are always pros and cons for different reasonable courses of action. One should gather data for each of the set of variables that are of concern or interest and determine weights among those to generate a final leading decision. For instance, when Warby Parker selected their first office location outside New York, they considered a large set of variables—Gallup's Well-being index, talent pool, cost of living, number and cost of flights to New York etc.—and ranked and weighted them as part of the final decision. Marissa Mayer (CEO of Yahoo!) tells a similar story when choosing among competing job offers and making her decision to work for Google [6].

A true data-driven organization will be doing at least one of these things, something forward-looking where data is a first class citizen.

OK, now we have an organization that has high quality data, skilled analysts who are engaged in these forward looking activities. Surely, that makes it data-driven!

Unfortunately, not necessarily. Like a tree falling in a forest with no-one to hear it, if analysts are putting out analyses but no-one takes notice, if they don't influence decision makers' decisions, which are still based on gut and opinion, it is not data-driven. Analytics has to inform and influence the influencers.

Dykes talks about this in terms of an "analytics value chain" (see Figure 1-3). Data have to drive reports which should lead to deeper dives and analysis. Those analyses

6 Bosker, B. 2011. Google Exec Marissa Mayer Explains Why There Aren't More Girl Geeks. http://www.huffingtonpost.com/2011/07/06/google-marissa-mayer-women-in-tech_n_891167.html.

have to get to the decision maker who incorporates those into her decision-making process. This step is key to being data-driven. An organization needs those data and that analysis to drive a decision that changes strategy or tactics and makes an ultimate impact to the organization in some manner. Technology and training can do the first part: enable analysts to run analyses and write up their findings. However, it is the *culture* that sets up the mindset and process to take notice of those findings, trust them, and act upon them.

Finally, we get to the crux of what being data-driven means. A data-driven organization will use the data as critical evidence to help inform and influence strategy. There will be an evidence-based culture in which data can be trusted, the analysis is highly relevant, informative, and is used to determine next steps.

Therein lies the challenge. If your organization is making gut decisions, how do you make a case for more data-driven decision making? It is not easy and it is not quick, so don't expect radical changes overnight, but everyone in the organization can contribute significantly to such an improvement. In this book we will examine a number of those ways and the culture that the organization should be driving to be more data-driven.

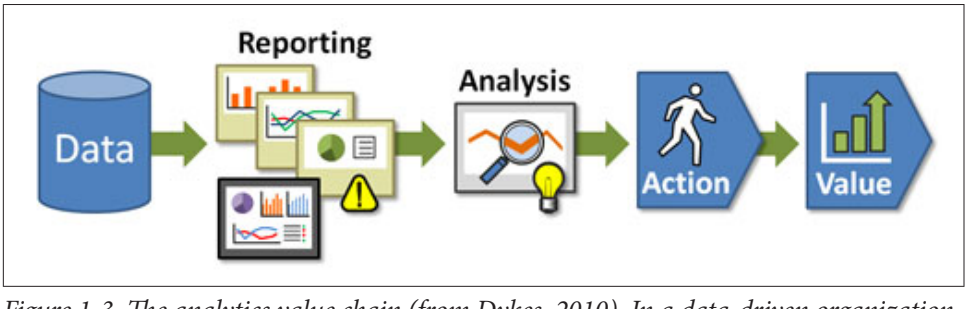


Figure 1-3. The analytics value chain (from Dykes, 2010). In a data-driven organization, the data feed reports, which stimulate deeper analysis. These are fed up to the decision makers who incorporate them into their decision-making process, influencing the direction that the company takes and providing value and impact. Figure from http://blogs.adobe.com/digitalmarketing/analytics/reporting-vs-analysis-whats-the-difference/.

Analytics Maturity

In 2009, Jim Davis, the senior vice-president and chief marketing officer of SAS Institute declared that there are eight levels of analytics[7]:

7 SAS. 2008. Eight Levels of Analytics. http://www.sas.com/news/sascom/analytics_levels.pdf.

1. **Standard reports**: What happened? When did it happen? *Example*: monthly financial reports.
2. **Ad hoc reports**: How many? How often? Where? *Example*: custom reports.
3. **Query drill down (or online analytical processing, OLAP)**: Where exactly is the problem? How do I find the answers? *Example*: data discovery about types of cell phone users and their calling behavior.
4. **Alerts**: When should I react? What actions are needed now? *Example*: CPU utilization mentioned earlier.
5. **Statistical analysis**: Why is this happening? What opportunities am I missing? *Example*: why are more bank customers refinancing their homes?
6. **Forecasting**: What if these trends continue? How much is needed? When will it be needed? *Example*: retailers can predict demand for products from store to store.
7. **Predictive modeling**: What will happen next? How will it affect my business? *Example*: casinos predict which VIP customers will be more interested in particular vacation packages.
8. **Optimization**: How do we do things better? What is the best decision for a complex problem? *Example*: what is best way to optimize IT infrastructure given multiple, conflicting business and resource constraints?

The ideas appear to form the basis of Figure 1-2 of Davenport and Harris's (2007) influential book *Competing on Analytics* [8]shown here as Figure 1-4.

8 Despite appearing two years ealier, Davenport and Harris' source is cited as "adapted from a graphic produced by SAS."

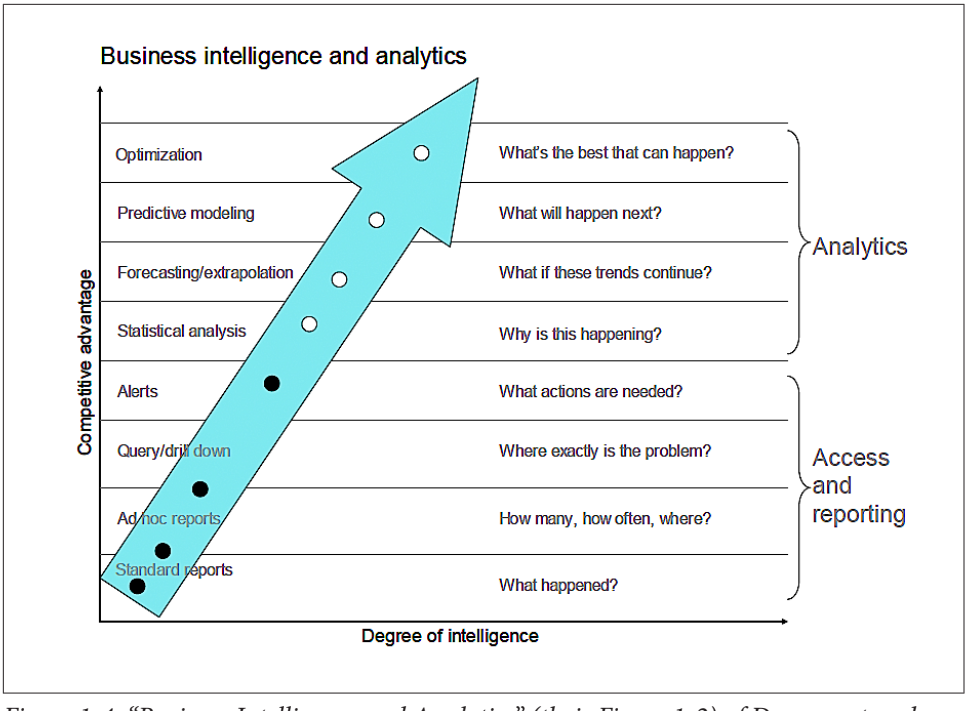


Figure 1-4. "Business Intelligence and Analytics" (their Figure 1-2) of Davenport and Harris (2007) Competing on Analytics. HBR Press.

(You can see that this is also where Davenport's framework and Table 1-2 is based. You can easily map the first four levels to the upper information row and the second set of four to the lower insight row of Table 1-2.)

I like the general concept and the labels. However, in the way that both Davis (2009) and Davenport and Harris (2007) present their ideas, especially with the big sweeping upward arrow, it is too easy to interpret this as a progression or some sort of hierarchy, kind of like a video game where you progress to each next level sequentially only having conquered the previous level.

This pseudo-progression is often labeled as analytics maturity. If you do a Google image search for "analytics maturity" you will see what I mean; that many BI vendors and practitioners present this as set of stepping stones with unidirectional arrows pointing from one level to the next. Analytics is not like that, it cuts across levels within an analysis and different parts of the organization can be engaged in analyses

of differing degrees of sophistication at any one time. Ron Shevlin[9] makes some great points:

> From a capabilities perspective, there's no reason why a firm couldn't forecast something likes sales ("level" 6) without knowing where exactly the "problem" with sales is ("level" 3)...How could I, as a manager, address the question of "what actions are needed now?" without some understanding of "what if these trends continue?" and "what will happen next?" ("levels" 6 and 7)?

The correct way, I think, to interpret this is to think of the *maximum* level that the organization is engaged in is positively correlated with the level of commitment, investment, and utility of analytics and, as Davenport and Harris argue, the analytical competitiveness. For instance, if you have an operations research team of Ph.D.s dedicated to optimizing your global supply chain, you are clearly heavily invested in data and analytics. If your organization only gets to alerts and query drilldowns, you have a lower investment and are less data-driven.

The underlying implication is that more sophisticated analytics are better, they make and organization more competitive. Is that true? In a fascinating study [10], MIT Sloan Management Review collaborated with IBM Institute for Business Value to survey 3000 managers and analysts across 30 industries about their use of and beliefs about the value of analytics.

One survey question asked about the organization's competitive position where the possible responses were:

1. Substantially outperform industry peers.
2. Somewhat outperforming industry peers.
3. On par with industry peers.
4. Somewhat or substantially underperforming industry peers.

Those organizations that chose answers 1 or 4 were deemed top and lower performers respectively. Interestingly, compared to lower performers, top performers were

- Five times more likely to use analytics.
- Three times more likely to be *sophisticated* analytics users.
- Two times more likely to use analytics to guide day-to-day operations.

---

9 Shevlin, R. 2009. The Eight Levels Of Analytics? http://snarketing2dot0.com/2009/10/27/the-eight-levels-of-analytics/.

10 Analytics: the New Path to Value. Research Report Fall 2010. http://sloanreview.mit.edu/article/big-data-analytics-and-the-path-from-insights-to-value/.

- Two times more likely to use analytics to guide future strategies.

There are certainly complicating factors in the methodology. There may be significant survivor bias and there will likely be correlation of top performers with organization size (we know that revenue of these organizations ranged from less than $500M to more than $10Bn). For instance, perhaps only larger, more successful organizations have the bandwidth and resources to develop sophisticated operations research departments that can develop and run supply chain simulation models. However, there was broad agreement that better and more sophisticated analytics drove business value.

The authors identified three levels of analytics capability: aspirational, experienced, and transformed. These are summarized in Table 1-3.

Table 1-3. Levels of analytics capability: aspirational, experienced, and transformed. Modified from http://sloanreview.mit.edu/article/big-data-analytics-and-the-path-from-insights-to-value/

	Aspirational	Experienced	Transformed
Use analytics to...	Justify actions	Guide actions	Prescribe actions
Use rigorous approaches to make decisions	Rarely	Sometimes	Mostly
Ability to capture, aggregate, and analyze or share information and insights	Limited	Moderate	High
Functional proficiency	• Finance, budgeting • Operations and production • Sales and marketing	• All aspirational functions • Strategy / biz-dev • Customer service • Product R&D	• All aspirational and experienced functions • Risk management • Customer experience • Work force planning • General management • Brand and marketing management

Compared to aspirational organizations, transformed organizations were:

- Four times more likely to capture information very well.
- Nine times more likely to aggregate information very well.
- Eight times more likely to analyze information very well.
- Ten times more likely to disseminate information and insights very well.

- 63% more likely to use a centralized analytics unit as the primary source of analytics (analytics organizational structures are covered in ???).

Again, there is a complicated tangle of cause and effect and biases here but there is an association between competitive advantage, relative to industry peers, and analytics sophistication.

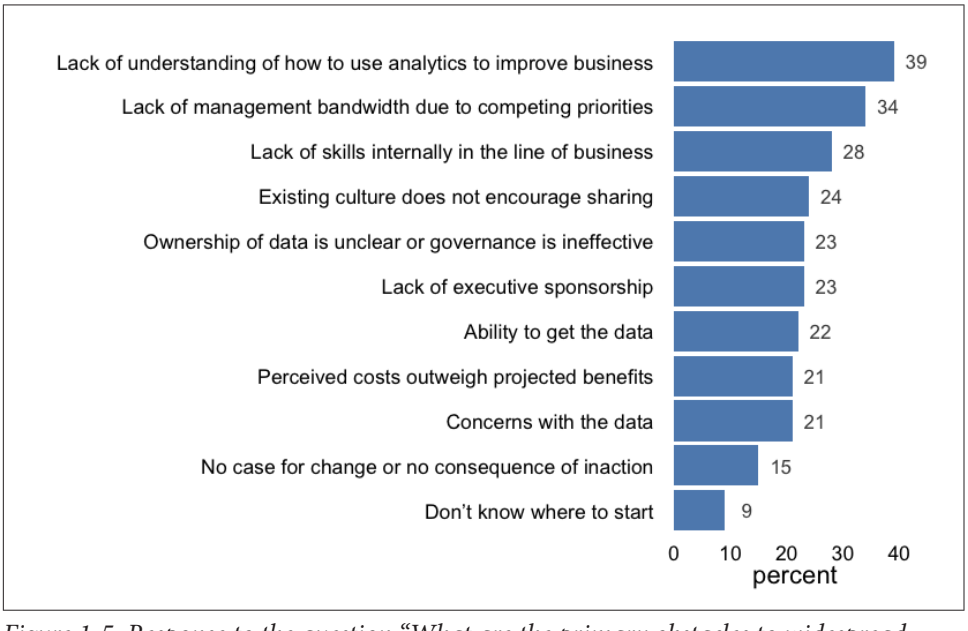


Figure 1-5. Response to the question "What are the primary obstacles to widespread adoption and use of information and analytics in your organization?"

So, what is stopping organizations adopting analytics widely? Two of the top three answers were lack of understanding of how to use analytics and lack of skills internally in the line of business (see Figure 1-5). Those are something that everyone, including every analyst, can help drive. For instance, analysts can help "level up" their skill set and can be more proactive about doing good work and communicating its value to their managers and to the business. They can do more research to dig up case studies of how other organizations tackled similar business problems through analytics. Managers of the data engineers can assign resource to data integration and quality so that data are trusted. Senior managers can promote or demand greater sharing of data and designate clearer ownership and stewardship of data, such as appointing a chief analytics officer or chief data officer (covered in ???). Everyone has a role.

Overview

Roughly speaking, this book is organized by thinking about that flow along that value chain. The first chapters cover data itself, in particular choosing the right data sources and ensuring that they are high quality and trustworthy. The next step in that chain is analysis. You need the right people with the right skills and tools to do good work, to generate impactful insights. I call this group the "analysts," deliberately using the term in its broadest sense to cover data analysts, data scientists, and other members of the analytics organization. I do that to be inclusive as I believe that everyone, from a junior data analyst fresh out of school to a rockstar data scientist has a role to play. I cover what makes a good analyst, how they can sharpen their skills, and also cover organizational aspects: how those analysts should be formed into teams and business units. The next few chapters cover the actual analytical work itself such as performing analysis, designing metrics, A/B testing, and story telling. I then proceed to the next step in the chain: making decisions with those analyses and insights. Here I address what makes decision making hard and how it can be improved.

Throughout all these chapters, there is a very strong message and theme: being data-driven is not just about data or the latest big data toolset, but it is *culture*. Culture is the dominant aspect that sets expectations of how far data are democratized, how they are used and viewed across the organization, and the resources and training invested in using data as a strategic asset. Thus, I draw all the lessons from the various steps in the value chain into a single culture chapter. One of the later chapters then discusses top-down data leadership and in particular, the roles of two relatively new C-suite positions: the chief data officer and the chief analytics officer. However, culture can be shaped and influenced from the bottom up too. Thus, throughout the book I directly address analysts and managers of analysts, highlighting what they can do to influence that culture and maximize their impact upon the organization. A true data-driven organization is a data democracy and has a large number of stakeholders throughout the organization who are vested in data and data quality, the best use of data to make fact-based decisions, and to leverage data for competitive advantage.

O'REILLY®

The Human Side of Postmortems

Managing Stress and Cognitive Biases

David Zwieback

The Human Side of Postmortems

by Dave Zwieback

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

May 2013: First Edition

Revision History for the First Edition:

2013-05-07: First release

2014-04-09: Second release

2015-03-24: Third release

ISBN: 978-1-449-36585-1

Table of Contents

Acknowledgements

The author greatfully acknowledges the contributions of the following individuals, whose corrections and ideas made this article vastly better: John Allspaw, Gene Kim, Mathias Meyer, Peter Miron, Alex Payne, James Turnbull, and John Willis.

What's Missing from Postmortem Investigations and Write-Ups?

How would you feel if you had to write a postmortem containing statements like these?

> "We were unable to resolve the outage as quickly as we would have hoped because our decision making was impacted by extreme stress."
>
> "We spent two hours repeatedly applying the fix that worked during the previous outage, only to find out that it made no difference in this one."
>
> "We did not communicate openly about an escalating outage that was caused by our botched deployment because we thought we were about to lose our jobs."

While these scenarios are entirely realistic, I challenge the reader to find many postmortem write-ups that even hint at these "human factors." A rare and notable exception might be Heroku's "Widespread Application Outage"[1] from the April 21, 2011, "absolute disaster" of an EC2 outage, which dryly notes:

> Once it became clear that this was going to be a lengthy outage, the Ops team instituted an emergency incident commander rotation of 8 hours per shift, keeping a fresh mind in charge of the situation at all time.

The absence of such statements from postmortem write-ups might be, in part, due to the social stigma associated with publicly acknowledging the contribution of human factors to outages. And yet, people dealing with outages are subject to physical exhaustion and psycho-

1. *http://bit.ly/KVKqB0*

logical stress and suffer from communication breakdowns, not to mention impaired reasoning due to a host of cognitive biases.

What *actually* happens during and after outages is this: from the time that an incident is detected, imperfect and incomplete information is uncovered in *nonlinear*, chaotic bursts; the full outage impact is not always apparent; the search for "root causes" often leads down multiple dead ends; and not all conditions can be immediately identified and remedied (which is often the reason for repeated outages).

The omission of human factors makes most postmortem write-ups a peculiar kind of docufiction. Often as long as novellas (see Amazon's 5,694-word take on the same outage discussed previously in "Summary of the April 21, 2011 EC2/RDS Service Disruption in the US East Region"[2]), they follow a predictable format of the Three Rs[3]:

- **Regret** — an acknowledgement of the impact of the outage and an apology.
- **Reason** — a *linear* outage timeline, from initial incident detection to resolution, including the so-called "root causes."
- **Remedy** — a list of remediation items to ensure that this particular outage won't repeat.

Worse than not being documented, human and organizational factors in outages may not be sufficiently considered during postmortems that are narrowly focused on the technology in complex systems. In this paper, I will cover two additions to outage investigations — stress and cognitive biases — that form the often-missing human side of postmortems. How do we recognize and mitigate their effects?

2. *http://amzn.to/jFdKAR*
3. McFarlan, Bill. *Drop the Pink Elephant: 15 Ways to Say What You Mean… and Mean What You Say*. Capstone, 2009.

Stress

What Is Stress?

Outages are stressful events. But what does *stress* actually mean, and what effects does it have on the people working to resolve an outage?

The term *stress* was first used by engineers in the context of stress and strain of different materials and was borrowed starting in the 1930s by social scientists studying the effects of physical and psychological stressors on humans[1]. We can distinguish between two types of stress: absolute and relative. Seeing a hungry tiger approaching will elicit a stress reaction — the fight-or-flight response — in most or all of us. This evolutionary survival mechanism helps us react to such absolute stressors quickly and automatically. In contrast, a sudden need to speak in front of a large group of people will stress out many of us, but the effect of this relative stressor would be less universal than that of confronting a dangerous animal.

More specifically, there are four relative stressors that induce a measurable stress response by the body:

1. A situation that is *interpreted* as novel.
2. A situation that is *interpreted* as unpredictable.
3. A *feeling* of a lack of control over a situation.

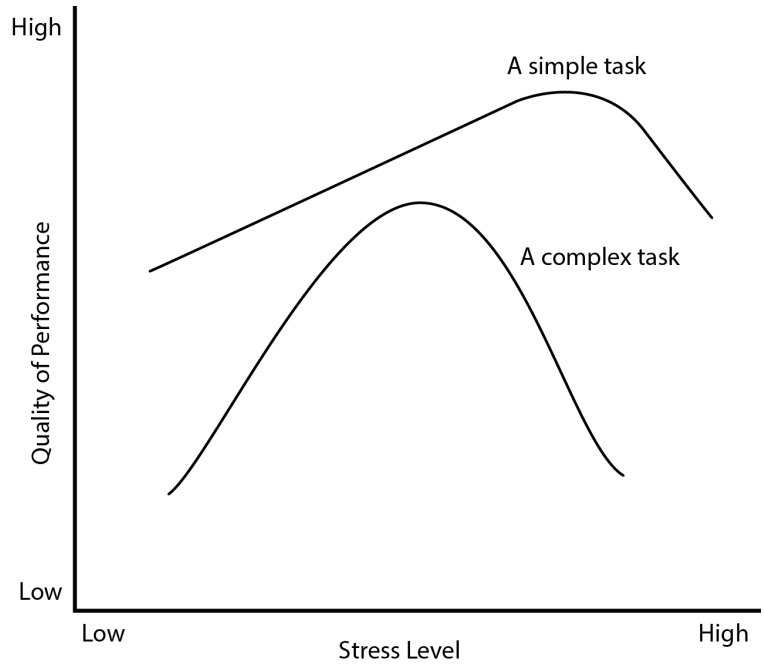
1. Lupien, Sonia J., F. Maheu, M. Tu, Al Fiocco, and T. E. Schramek. "The effects of stress and stress hormones on human cognition: implications for the field of brain and cognition." Brain and Cognition 65, no. 3 (2007): 209-237.

4. A situation where one can be judged negatively by others (the "social evaluative threat").

While most outages are not life-or-death matters, they still contain combinations of most (or all) of the above stressors and will therefore have an impact on the people working to resolve an outage.

Performance under Stress

In 1908, the psychologists Robert Yerkes and John Dodson established a relationship between stress and performance. Although what is now known as the Yerkes-Dodson law was based on a less-than-humane experiment with a few dozen mice, subsequent research confirmed that it was "valid in an extraordinarily wide range of situations."[2]



Not all stress is bad. For instance, as you can see from the diagram above, low levels of stress are actually associated with low levels of performance. For example, it's unlikely that one will do one's best work of the day right after waking up, without taking steps to shake off the

2. Kahneman, Daniel. "Attention and effort." (1973).

grogginess (e.g., coffee, a morning run, and there's nothing like reading a heated discussion on Hacker News to get the heart rate up).

As stress increases, so does performance, at least for some time. This is the reason that a coach gives a rallying pep talk before an important sports event — a much-parodied movie cliché that can nonetheless improve team performance. Athletes are also often seen purposefully putting themselves in higher stress situations before competitions (for instance by playing loud music or warming up vigorously) in order to improve focus, motivation, and performance.

While the Yerkes-Dodson law applies universally, individuals exhibit a wide spectrum of stress tolerance. Some people are extraordinarily resilient to high levels of stress, and some of them naturally gravitate toward high-stress professions that involve firefighting (both the literal and figurative kinds). However, there is an inflection point for each individual after which additional stress will cause performance to deteriorate due to impaired attention and reduced ability to make sound decisions. The length of time that one is subject to stress also impacts the extent of its effects: playing Metallica's "Enter Sandman" at top volume might initially improve performance, but continued exposure will eventually weaken it. (Notably, this song has been used to put Guantánamo Bay detainees under extreme stress during interrogations[3].)

Simple vs. Complex Tasks

An important part of the Yerkes-Dodson law that is often overlooked is that simple tasks are much more resilient to the effects of stress than complex ones. That is, in addition to individual differences in stress resilience, the impact of stress on performance is also related to the difficulty of the task.

One way to think about "simple" tasks is that they are well-learned, practiced, and relatively effortless. For instance, one will have little difficulty recalling the capital of France, regardless of whether one is in a low- or high-stress situation. In contrast, "complex" tasks (like troubleshooting outages) are likely to be novel, unpredictable, or perceived as outside one's control. That is, complex tasks are likely to be subject to three of the four relative stressors mentioned above.

3. Stafford Smith, Clive. The Guardian, "Welcome to *the disco*." *http://bit.ly/oE3UM*.

With practice, complex tasks can become simpler. For instance, driving is initially a very complex task. Because learning to drive requires constant and effortful attention, one is unlikely to be playing "Harlem Shake" at top volume or casually chatting with friends at the same time. As we become more experienced, driving becomes more automatic and much less effortful, though we might still turn down the radio volume or pause our conversations when merging into heavy traffic. The good news is that increased experience in a particular task can make its performance more resilient to the effects of stress.

Stress Surface, Defined

The difficulty, of course, is finding precisely the point of an individual's optimal performance as it relates to stress during an outage. A precise measurement is impractical, since it would involve ascertaining the difficulty of the task, type and duration of stress, and would also have to account for individual differences in stress response.

A more pragmatic approach is to estimate the *potential* impact that stress can have on the outcome of an outage. To enable this, I'm introducing the concept of "stress surface," which measures the perception of the four relative stressors during an outage: the novelty of the situation, its unpredictability, lack of control, and social evaluative threat. These four stressors are selected because they are present during most outages, are known to cause a stress response by the body, and therefore have the potential to impact performance.

Stress surface is similar to the computer security concept of "attack surface" — a measure of the collection of ways in which an attacker can damage a system[4]. Very simply, an outage with a larger stress surface is more susceptible to the effects of stress than that with a smaller stress surface. As a result, we can use stress surface to compare the potential impact of stress on different outages as well as assess the impact of efforts to reduce stress surface over time.

To measure stress surface, we use a modified Perceived Stress Scale, the "most widely used psychological instrument for measuring the perception of stress":[5]

4. Manadhata, Pratyusa K. "Attack Surface Measurement." *http://bit.ly/10niL47*.

5. Cohen, Sheldon. "Perceived Stress Scale." *http://bit.ly/wmXLU8*.

The questions in this scale ask you about your feelings and thoughts during the outage. In each case, you will be asked to indicate how often you felt or thought a certain way.

0 = Never 1 = Almost Never 2 = Sometimes 3 = Fairly Often 4 = Very Often

During the outage, how often have you felt or thought that:

1. The situation was novel or unusual?
2. The situation was unpredictable?
3. You were unable to control the situation?
4. Others could judge your actions negatively?

We administer the above questionnaire as soon as possible after the completion of an outage. To prevent groupthink, all participants of the postmortem should complete the questionnaire independently. The overall stress surface score for each outage is obtained by summing the scores for all responses. A standard deviation should also be computed for the score to indicate the variance in responses.

Why measure stress surface? Knowing the stress surface score and asking questions like "What made this outage feel so unpredictable?" opens the door to understanding the effects of stress in real-world situations. Furthermore, one can gather data about the relationship of stress to the length of outages and determine if any particular dimension of the stress surface (for example, the threat of being negatively judged) remains stable between various outages. Most important, stress surface allows us to measure the results of steps taken to mitigate the effects of stress over time.

Reducing the Stress Surface

Two effective ways to reduce the stress surface of an outage are training and postmortem analyses. Specifically, conducting realistic game day exercises; regular disaster recovery tests; or, if operating in Amazon Web Services (AWS), surprise attacks of the Netflix Simian Army[6] — all followed by postmortem investigations — are effective in making

6. *http://nflx.it/q6fVuL*

outages less novel as well as exposing latent failure conditions. Moreover, developing so-called "muscle memory" from handling many outages (including practicing critical communication skills) can reduce the perceived complexity of tasks, making their performance more resilient to the effects of stress.

There has also been some promising research into Decision Support Systems (DSS), which have been used to improve decision making under stress in military and financial applications. In one case, researchers attached biometric monitors to bank traders, which alerted them when decision making was likely to be compromised due to high stress (measured by the stability of the frequency and shape of the heart rate waveform[7]). While DSS technology matures, organizations with awareness of the effects of stress on performance can take simple stress mitigation steps, for instance, by insisting on a "rotation of 8 hours per shift" during lengthy outages.

Why Postmortems Should Be Blameless

Unfortunately, these stress surface reduction steps do not address the effects of social evaluative threat in meaningful ways. That is especially troubling because, in my early investigations into stress surface, the component related to being negatively judged appears most stable between different outages and engineers.

Evaluative threat is social in nature — it involves both the organization's ways of dealing with failure (e.g., the extent to which blame and shame are part of the culture) and the individual's ability to cope with it. We should not dismiss the extent to which this stressor affects performance: several surveys have found that Americans are more afraid of public speaking, which is a classic example of social evaluative threat, than death[8]. Organizations where postmortems are far from blameless and where being "the root cause" of an outage could result in a demotion or getting fired will certainly have larger stress surfaces.

The most effective way of mitigating the effects of social evaluative stress is to emphasize the blameless nature of postmortems. What does

7. Martínez Fernández, Javier, Juan Carlos Augusto, Ralf Seepold, and Natividad Martinez Madrid. "Sensors in trading process: A Stress — Aware Trader." InIntelligent Solutions in Embedded Systems (WISES), 2010 8th Workshop, pp. 17-22. IEEE, 2010.

8. Garber, Richard I. "America's Number One Fear: Public Speaking - that 1993 Bruskin-Goldring Survey." Last modified May 19, 2011. *http://bit.ly/11KpT77*.

"blameless" actually mean? Very simply, *your organization must continually affirm that individuals are never the "root cause" of outages*. This can be counterintuitive for engineers, who can be quick to take responsibility for "causing" the failure or to pin it on someone else. In reality, blame is a shortcut, an intuitive jump to an incorrect conclusion, and a symptom of not going deeply enough in the postmortem investigation to identify the real conditions that enabled the failure, conditions that will likely do so again until fully remediated.

Making the effort to become more accepting of failure at an organizational level, and more specifically making postmortems "blameless," is not a new-age feel-good measure done intuitively in "evolved" organizations. It is rooted in the understanding of the real conditions of failure in complex systems and a concrete way to improve performance during outages by reducing their stress surface.

The Limits of Stress Reduction

Of course, no amount of training or experience can reduce the stress surface to zero — outages will continue to surprise (and to some extent delight) in novel, unpredictable ways. A true mark of an expert is a realistic and humble assessment of the limitations of experience and the extent to which control over complex systems is actually possible. In contrast, less mature engineers tend to develop overconfidence in their own abilities after some initial success and familiarly with systems. This is not endemic to engineers: despite overwhelming evidence that inexperience is one of the main causes of accidents in young drivers, they consistently fail to judge the extent of their own inexperience and how it affects their safety[9]. We'll cover overconfidence and other biases in more detail later in this paper.

Caveats of Stress Surface Measurements

In a poll of 2,387 U.S. residents, the mean male and female Perceived Stress Scale scores (12.1 and 13.7, respectively) had fairly high stan-

9. Ginsburg, Kenneth R., Flaura K. Winston, Teresa M. Senserrick, Felipe García-España, Sara Kinsman, D. Alex Quistberg, James G. Ross, and Michael R. Elliott. "National young-driver survey: teen perspective and experience with factors that affect driving safety." Pediatrics 121, no. 5 (2008): e1391-e1403.

dard deviations (5.9 and 6.6, respectively)[10]. We can expect a similarly high variance in stress surface measurements, in part due to the individual differences in perception of stress.

We should also remember that stress surface scores are based on a *memory* of feelings and thoughts during a stressful event. There are many conditions that could influence the ability of individuals to faithfully recall their experiences, including the duration of time that has passed since the event as well as the severity of stress they experienced during an outage. Furthermore, our recollections are likely colored by hindsight bias, which is our tendency to remember things as more obvious than they appeared at the time of the outage.

Finally, stress surface measurements in smaller teams may be subject to the Law of Small Numbers. As Daniel Kahneman warns in *Thinking, Fast and Slow*:[11]

- The exaggerated faith in small samples is only one example of a more general illusion — we pay more attention to the content of messages than to information about their reliability, and as a result, end up with a view of the world around us that is simpler and more coherent than the data justify. Jumping to conclusions is a safer sport in the world of our imagination than it is in reality.
- Statistics produce many observations that appear to beg for causal explanations but do not lend themselves to such explanations. Many facts of the world are due to chance, including accidents of sampling. Causal explanations of chance events are inevitably wrong.

Nevertheless, obtaining the stress surface score for each outage is an effective way to frame the discussion of the effects of stress, including identifying ways they can be mitigated.

10. Cohen, Sheldon. "Perceived Stress Scale." *http://bit.ly/wmXLU8*.

11. Kahneman, Daniel. Thinking, fast and slow. Farrar, Straus and Giroux, 2011.

Cognitive Biases

The Benefits and Pitfalls of Intuitive and Analytical Thinking

To further quote Kahneman, "the law of small numbers is a manifestation of a general bias that favors certainty over doubt." What other biases affect people working with complex systems? And how can they be overcome?

To begin our discussion of cognitive biases, we should introduce the theory of two different systems (or types) of thinking. Perhaps the best way to do it is through examples:

> What is 2+2?
>
> What is the capital of France?

Did you notice that in the above examples, you arrived at the answers (4 and Paris, respectively) quickly, automatically, and without effort? In fact, there was almost nothing you could do to stop the answers from appearing in your mind. This fast, intuitive, and largely automatic thinking is known as System 1 thinking. This type of thinking is largely based on associative memory and experience. This is the thinking celebrated in Malcom Gladwell's 2005 book *Blink: The Power of Thinking Without Thinking*. A famous example from *Blink* is that of the Getty kouros, a statue bought by the Getty Museum. Despite the statue's credible documentation, expert archeologists identified it as fake, seemingly at a glance and despite their inability to specify the exact reasons.

Now, let's illustrate the other (System 2) thinking:

> What is 367 x 108?

Unless you've memorized the answer previously, or have made a career of performing feats of mental math, it did not automatically come to mind and it took some time to calculate (the answer is 39,636). To do so, you used your System 2 thinking, which, compared to System 1 thinking, is slower, non-automatic, and effortful. In fact, it uses more energy (glucose) than System 1 thinking. As a result, we spend much of our time relying on the more energy-efficient System 1 thinking. System 1 is also where we fall under stress.

The two systems of thinking are not separate, and there's no reason to look down on the lowly System 1 thinking. This is the thinking that allows us to do marvelous things like drive a car and listen to music or hold a conversation at the same time, to quickly determine if our partner is upset after the first few moments of a phone conversation, or enables an experienced firefighter to save his men by pulling them out of a dangerous fire based on a sudden "gut feeling" that something is wrong[1].

However, System 1 thinking has two major shortcomings. First, it is rooted in memory and experience. Unless we've trained for years as archeologists, and have looked at literally thousands of archeological artifacts, we won't be able to "take in" a new artifact and quickly determine its authenticity. Similarly, it is the many years of seeing complex systems function and fail that allows experienced operations people to quickly identify and deal with conditions of an outage. The second shortcoming of System 1 thinking is that it is prone to making systematic mistakes, which are called cognitive biases. Our preference for System 1 thinking, especially in stressful situations, can increase the effects of cognitive biases during outages.

Jumping to Conclusions

Consider this question:

> If it takes 5 machines 5 minutes to make 5 widgets, how long would it take 100 machines to make 100 widgets?

This is one of the three questions on a Cognitive Reflection Test — questions that were selected because they reliably evoke an immediate

1. Gladwell, Malcolm. Blink: The power of thinking without thinking. Back Bay Books, 2007.

and *incorrect* answer[2]. You are in good company if your quick and intuitive answer is "100 minutes" — 90% of participants of an experiment involving Princeton students answered at least one of the three CRT questions incorrectly. Still, the unintuitive and correct answer is "5 minutes."

What's going on here? As we've seen, System 1 thinking quickly and efficiently provides what you might call a first approximation assessment of a situation, or the effortlessly intuitive answer to the question above. And in the vast majority of cases, System 1 thinking functions superbly. For instance, being able to quickly spot something that looks like a leopard is an important function of System 1. When System 1 produces a mistake — if, for instance, what looks like a leopard turns out to be an old lady in a leopard-print coat — from an evolutionary point of view, it may be vastly better to be wrong while quickly running away than to be mauled by a hungry leopard while taking the time to thoroughly analyze the potential attacker. That is, unless you, as a result, quickly run into crosstown traffic, in which case it would have been far better to slow down and actually evaluate the probability of meeting a leopard in midtown Manhattan!

System 1 is expert at quickly jumping to conclusions. It does so by employing mental shortcuts — heuristics — "which reduce the complex tasks of assessing probabilities and predicting values to simpler judgmental operations. In general, these heuristics are quite useful, but sometimes they lead to severe and systematic errors,"[3] otherwise known as cognitive biases.

A Small Selection of Biases Present in Complex System Outages and Postmortems

There are more than 100 cognitive biases listed in Wikipedia[4], and Daniel Kahneman's epic-yet-accessible treatment of the subject (*Thinking, Fast and Slow*) weighs in at more than 500 pages.

2. Kahneman, Daniel. Thinking, fast and slow. Farrar, Straus and Giroux, 2011.
3. Tversky, Amos, and Daniel Kahneman. Judgment under uncertainty: Heuristics and biases. Springer Netherlands, 1975.
4. *http://bit.ly/985JMi*

Both the number of biases and our understanding of them is growing, as they have been the subject of considerable research since they were first identified by Kahneman and his research partner Amos Tversky in the early 1970s.

The following discussion will give the reader familiarity with some of the more "classic" biases that are usually present during outages and postmortems.

Hindsight Bias

In early 1999, the co-founders of Google, having raised a mere $100,000 to date, attempted to sell it to more established search companies Yahoo! and Excite so they could return to their graduate studies. [5] What is your estimate of the asking price?

Clearly, the time at which the above question was posed would have an effect on your answer. If you were asked in early 1999, would you have guessed that $1 million was a reasonable estimate? Had you been asked the same question in 2012, would you have thought that Google could have been as valuable to Yahoo! as Instagram was to Facebook, and therefore worth $1 billion (or $725 million, adjusted for inflation)?

Moreover, given what we know about Google's current valuation (more than $266 billion) as well as the dwindling fortunes of Yahoo! and Excite, it appears absolutely clear that these companies missed a huge opportunity by not purchasing Google for the actual asking price of $1 million (or less).

The above statement is an example of several biases in action, most prominently the hindsight bias. This bias affects not just casual observers, but professionals as well. For example, here is how Paul Graham, an investor at the Y Combinator startup incubator, views the same situation:

> Google's founders were willing to sell early on. They just wanted more than acquirers were willing to pay … Tip for acquirers: when a startup turns you down, consider raising your offer, *because there's a good chance the outrageous price they want will later seem a bargain* [6].

5. Siegler, MG. TechCrunch. "When Google Wanted To Sell To Excite For Under $1 Million — And They Passed". *http://tcrn.ch/ctS4eM*.

6. Graham, Paul. "Why There Aren't More Googles." *http://bit.ly/z3zoX*.

In retrospect, buying Google for $1 million in 1999 certainly looks like a fantastic investment. However, Google's success was far from certain in 1999, and given that 75% of startups fail[7], was it really as good a chance as Graham seems to think?

During postmortems we evaluate what happened during an outage with the benefit of currently available information, i.e., hindsight. As we aim to identify the conditions that were necessary and sufficient for an outage to occur, we often uncover things that could have prevented or shortened the outage. We hear statements like "You shouldn't have made the change without backing up the system first" or "I don't know how I overlooked this obvious step" from solemn postmortem participants. Except that these things were as far from obvious during the outage as Google's 2013 valuation was in 1999!

Outcome Bias

When the results of an outage are especially bad, hindsight bias is often accompanied by outcome bias, which is a major contributor to the "blame game" during postmortems. Because of hindsight bias, we first make the mistake of thinking that the correct steps to prevent or shorten an outage are equally obvious before, during, and after the outage. Then, under the influence of outcome bias, we judge the quality of the actions or decisions that contributed to the outage in proportion to how "bad" the outage was. The worse the outage, the more we tend to blame the human committing the error — starting with overlooking information due to "a lack of training," and quickly escalating to the more nefarious "carelessness," "irresponsibility" and "negligence." People become "root causes" of failure, and therefore something that must be remediated.

The combined effects of hindsight and outcome bias are staggering:

> Based on an actual legal case, students in California were asked whether the city of Duluth, Minnesota, should have shouldered the considerable cost of hiring a full-time bridge monitor to protect against the risk that debris might get caught and block the free flow of water. One group was shown only the evidence available at the time of the city's decision; 24% of these people felt that Duluth should take on the expense of hiring a flood monitor. The second group was informed that debris had blocked the river, causing major flood damage;

7. Xavier, Jon. Silicon Valley Business Journal, "75% of startups fail, but it's no biggie." *http://bit.ly/QGUSdC*.

> 56% of these people said the city should have hired the monitor, *although they had been explicitly instructed not to let hindsight distort their judgment* [8].

Outcome bias is also implicated in the way we perceive risky actions that appear to have positive effects. As David Woods, Sidney Dekker and others point out, "good decision processes can lead to bad outcomes and good outcomes may still occur despite poor decisions"[9]. For example, if an engineer makes changes to a system without having a reliable backup and this leads to an outage, outcome bias will help us quickly (and incorrectly) see these behaviors as careless, irresponsible, and even negligent. However, if no outage occurred, or if the same objectively risky action resulted in a positive outcome like meeting a deadline, the action would be perceived as far less risky, and the person who took it might even be celebrated as a visionary hero. At the organizational level, there is a real danger that unnecessarily risky behaviors would be overlooked or, worse yet, rewarded.

Availability Bias

Residents of the Northeast United States experience electricity outages fairly frequently. While most power outages are brief and localized, there have been several massive ones, including the blackout of August 14-15, 2003[10]. Because of the relative frequency of such outages, and the disproportionate attention they receive in the media, many households have gasoline-powered backup generators with enough fuel to last a few hours. In late October 2012, in addition to lengthy power outages, Hurricane Sandy brought severe fuel shortages that lasted for more than a week. Very few households were prepared for an extended power outage *and* a gasoline shortage by owning backup generators *and* stockpiling fuel.

This is a demonstration of the effects of the availability bias (also known as the recency bias), which causes us to overestimate (sometimes drastically) the probability of events that are easier to recall and underestimate that of events that do not easily come to mind. For instance, tornadoes (which are, again, heavily covered by the media)

8. Kahneman, Daniel. Thinking, fast and slow. Farrar, Straus and Giroux, 2011.
9. Woods, David D., Sidney Dekker, Richard Cook, Leila Johannesen, and N. B. Sarter. "Behind human error." (2009): 235.
10. *http://en.wikipedia.org/wiki/Northeast_blackout_of_2003*

are often perceived to cause more deaths than asthma, while in reality asthma causes 20 times more deaths.[11]

In the case of Hurricane Sandy, since the median age of the U.S. population is 37, the last time fuel shortages were at the top of the news (in 1973-74 and 1979-80) was before about half of the U.S. population was born, so it's easy to see how most people did not think to prepare for this eventuality. Of course, the hindsight bias makes it obvious that such preparations were necessary.

The availability bias impacts outages and postmortems in several ways. First, in preparing for future outages or mitigating effects of past outages, we tend to consider scenarios that appear more likely, but are, in fact, only easier to remember either because of the attention they received or because they occurred recently. For instance, due to its severity, many organizations utilizing AWS vividly remember the April 21, 2011, "service disruption" mentioned previously and have taken steps to reduce their reliance on the Elastic Block Store (EBS), the network storage technology at the heart of the lengthy outage. While they would have fared better during the October 22, 2012, "service event" also involving EBS, these preparations would have done little to reduce the impact of the December 24, 2012, outage, which affected heavy users of the Elastic Load Balancing (ELB) service, like Netflix.

Furthermore, especially under stress, we often fall back to familiar responses from prior outages, which is another manifestation of the availability bias. If rebooting the server worked the last N times, we are likely to try that again, especially if the initial troubleshooting offers no competing narratives. In general, not recognizing the differences between outages could actually make the situation worse.

Although much progress has been made in standardizing system components and configurations, outages are still like snowflakes, gloriously unique. Most outages are independent events, which means that past outages have no effect on the probability of future outages. In other words, while experience with previous outages is important, it can only go so far.

11. Kahneman, Daniel. Thinking, fast and slow. Farrar, Straus and Giroux, 2011.

Other Biases and Misunderstandings of Probability and Statistics

Most of us are terrible at *intuitively* grasping probabilities of events. For instance, we often confuse independent events (e.g., the probability of getting "heads" in a coin toss remains 50% regardless of the number of tosses) from dependent ones (e.g., the probability of picking a marble of a particular color changes as marbles are removed from a bag). This sometimes manifests as *sunk cost bias*, for example, when engineers are unwilling to try a different approach to solving a problem even though a substantial investment in a particular approach hasn't yielded the desired results. In fact, they are likely to exclaim "I almost have it working!" and further escalate their commitment to the non-working approach. This can be made worse by the *confirmation bias*, which compels us to search for or interpret information in a way that confirms our preconceptions.

At other times, intuitive errors in understanding of statistics result in finding illusory correlations (or worse, causation) between uncorrelated events — e.g., "every outage that Jim participates in takes longer to resolve, therefore the length of outages must have some relation to Jim." Similarly, because large outages are relatively rare, we can become biased due to the Law of Small Numbers — e.g., "this outage is likely to look like the last outage."

Finally, we are often overly confident in our decision-making abilities. This overconfidence bias manifests most clearly and dangerously when two nations are about to go to war, and their estimates of winning often sum to greater than 100% (i.e., "both think they have more than a 50% chance of winning"). Similarly, the positive "can do" attitude on display during outages is a symptom of overconfidence in our abilities to control the situation over which, in reality, we have little or no control (think: public cloud). There's certainly nothing wrong with maintaining a positive attitude during a stressful event, but it's worth keeping in mind that confidence is nothing but a feeling that is "determined mostly by the coherence of the story and by the ease with which it comes to mind, even when the evidence for the story is sparse and unreliable"[12].

12. Kahneman, Daniel. New York Times, "Don't Blink! The Hazards of Confidence." *http://www.nytimes.com/2011/10/23/magazine/dont-blink-the-hazards-of-confidence.html*.

Reducing the Effects of Cognitive Biases, or "How Do You Know That?"

Cognitive biases are a function of System 1 thinking. This is the thinking that produces quick, efficient, effortless, and intuitive judgments, which are good enough in most cases. But this is also the thinking that is adept at maintaining cognitive ease, which can lead to mistakes due to cognitive biases. The way that we can reduce the effects of cognitive biases is by engaging System 2 thinking in an effortful way. Even so:

> biases cannot always be avoided because System 2 may have no clue to the error … The best we can do is a compromise: learn to recognize situations in which mistakes are likely and try harder to avoid significant mistakes when the stakes are high[13].

We've discussed the effects of stress on performance, and we should emphasize again that we tend to slip into System 1 thinking under stress. This certainly increases the chances of mistakes that result from cognitive biases during and after outages. So what can we do to invoke System 2 thinking, which is less prone to cognitive biases, when we need it most?

We don't typically have the luxury of knowing when our actions might become conditions for an outage or when an outage may turn out to be especially widespread. However, before working on critical or fragile systems — or, in general, before starting work on large projects — we can use a technique developed by Gary Klein called the PreMortem. In this exercise, we imagine that our work has resulted in a spectacular and total fiasco, and "generate plausible reasons for the project's failure"[14]. Discussing cognitive biases in PreMortem exercises will help improve their recognition — and reduce their effects — during stressful events.

It's often easier to recognize other people's mistakes than our own. Working in groups and openly asking the following questions can illuminate people's quick judgments and cognitive biases at work:

> How is this outage different from previous outages?
>
> What is the relationship between these two pieces of information — causation, correlation, or neither?

13. Kahneman, Daniel. Thinking, fast and slow. Farrar, Straus and Giroux, 2011.
14. Klein, Gary. Harvard Business Review. "Performing a Project Premortem." *http://hbr.org/2007/09/performing-a-project-premortem/ar/1*

What evidence do we have to support this explanation of events?

Can there be a different explanation for this event?

What is the risk of this action? (Or, what could possibly go wrong?)

Edward Tufte, who's been helping the world find meaning in ever-increasing volumes of data for more than 30 years, suggests we view evidence (e.g., during an outage) through what he calls the "thinking eye," with:

> bright-eyed observing curiosity. And then what follows after that is reasoning about what one sees and asking: what's going on here? And in that reasoning, intensely, it involves also a skepticism about one's own understanding. The thinking eye must always ask: How do I know that? *That's probably the most powerful question of all time. How do you know that?*" [15]

15. Tufte, Edward. "Edward Tufte Wants You to See Better." Talk of the Nation, by Flora Lichtman. *http://www.npr.org/2013/01/18/169708761/edward-tufte-wants-you-to-see-better*.

Mindful Ops

Relative stressors and cognitive biases are both mental phenomena — thoughts and feelings — which nonetheless have concrete effects on our physical world, whether it is the health of operations people or the length and severity of outages. The best way to work with mental phenomena is through mindfulness. Mindfulness has two components:

> The first component involves the self-regulation of attention so that it is maintained on immediate experience, thereby allowing for increased recognition of mental events in the present moment. The second component involves adopting a particular orientation toward one's experiences in the present moment, an orientation that is characterized by curiosity, openness, and acceptance.[1]

One of the challenges with mitigating the effects of stress is the variance in individual responses to it. For instance, there is no known method to objectively determine the level of social evaluative threat that is harmful for a particular individual. Measuring stress surface, vital signs or stress hormone levels are, at best, proxies for — and approximations of — the real effects of stress. However, by practicing mindfulness, an individual can learn to recognize when they're experiencing (subjectively) harmful levels of stress and take simple corrective actions (e.g., take a break or ask for a second opinion in a high-risk situation). Mindfulness-Based Stress Reduction (MBSR) — a "meditation program created in 1979 from the effort to integrate Bud-

1. Bishop, Scott R., Mark Lau, Shauna Shapiro, Linda Carlson, Nicole D. Anderson, James Carmody, Zindel V. Segal et al. "Mindfulness: A proposed operational definition." Clinical psychology: Science and practice 11, no. 3 (2004): 230-241.

dhist mindfulness meditation with contemporary clinical and psychological practice" — is known to significantly reduce stress[2].

We can similarly mitigate the effects of cognitive biases through mindfulness — we can become aware of when we're jumping to conclusions and purposefully slow down to engage our analytical System 2 thinking.

The practice of mindfulness requires some effort, but is also simple, free, and without negative side effects. As we've seen, increased mindfulness — Mindful Ops — can reduce the effects of stress and cognitive biases, ultimately help us build more resilient systems and teams, and reduce the duration and severity of outages.

2. Chiesa, Alberto, and Alessandro Serretti. "Mindfulness-based stress reduction for stress management in healthy people: a review and meta-analysis." The journal of alternative and complementary medicine 15, no. 5 (2009): 593-600.

Author's Note

Meditation and mindfulness are huge subjects that we've barely begun to explore in this paper. I sincerely encourage the reader to investigate and experience their benefits in their work and life. The works of Thich Nhat Hanh, Jon Kabat-Zinn, Matthieu Ricard, or Sharon Salzberg (among others) are great places to get started.

About the Author

Dave Zwieback has been managing large-scale mission-critical infrastructure and teams for 17 years. He is the CTO of Lotus Outreach. He was previously the head of infrastructure at Knewton, managed UNIX Engineering at D.E. Shaw & Co., and managed enterprise monitoring tools at Morgan Stanley. He also ran an infrastructure architecture consultancy for seven years. Follow Dave *@mindweather* or on his website, *mindweather.com*.

THE **LEAN** SERIES ERIC RIES, SERIES EDITOR

Jez Humble, Joanne Molesky & Barry O'Reilly

LEAN ENTERPRISE

How High Performance Organizations Innovate at Scale

O'REILLY®

CHAPTER 15

Start Where You Are

> If you do something and it turns out pretty good, then you should go do something else wonderful, not dwell on it for too long. Just figure out what's next.
>
> Steve Jobs

> A year from now you will wish you had started today.
>
> Karen Lamb

Our goal with this book is to inspire you to envision an alternative future for large organizations. A future that puts employees, customers, and products at the heart of its strategy. A future where a renewed culture and environment enable the organization to adapt rapidly to changing market demands.

We have shared stories and lessons learned from a diverse set of organizations with varied backgrounds and circumstances to highlight that even in complex environments, you can thrive and address the most challenging problems. However, the path to success is not likely to be linear, with defined instructions, milestones, and KPIs. Organizations needed to get comfortable moving forward with uncertainty and imperfect information, while learning, adjusting, and developing their people along the way.

The biggest barrier to success in changing the way you work is a conviction that your organization is too big or bureaucratic to change, or that your special context prevents adopting the particular practices we discuss. Always remember that each person, team, and business that started this journey was unsure of what paths to take and how it would end. The only accepted truth was that if they failed to take action, a more certain, negative ending lay ahead.

Principles of Organizational Change

All change is risky, particularly organizational change which inherently involves cultural change—the hardest change of all, since you are playing with the forces that give the organization its identity. We are still amazed when leaders plan "organizational change" programs that they expect to complete in months. Such programs fail to recognize that turning innovation or change into an *event* rather than part of our daily work can never produce significant or lasting results. Periodically funding a new change program in response to current issues, leadership changes, or market trends without instilling a culture of experimentation will only achieve short-term incremental change, if any at all (Figure 15-1). Organizations will quickly slip back to their previous state. Instead, we must create a culture of continuous improvement through the deliberate, ongoing practice of everyone in the organization.

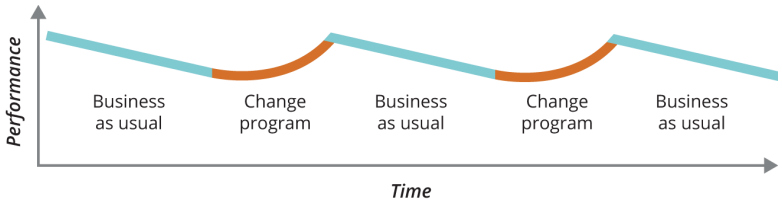


Figure 15-1. *The reality of "event-based" change programs*

If your organization is waiting for an event to stimulate change, you're already in trouble. In the current environment and competitive economy, a sense of urgency should be a permanent state. Survival anxiety always exists in leading organizations, as we describe in Chapter 11. However, as Schein noted, using it as a motivator for ongoing change is ineffective. The only path to a culture of continuous improvement is to create an environment where learning new skills and getting better at what we do is considered valuable in its own right and is supported by management and leadership, thus reducing learning anxiety. We can use the Improvement Kata presented in Chapter 6 to create this culture and drive continuous improvement (Figure 15-2).

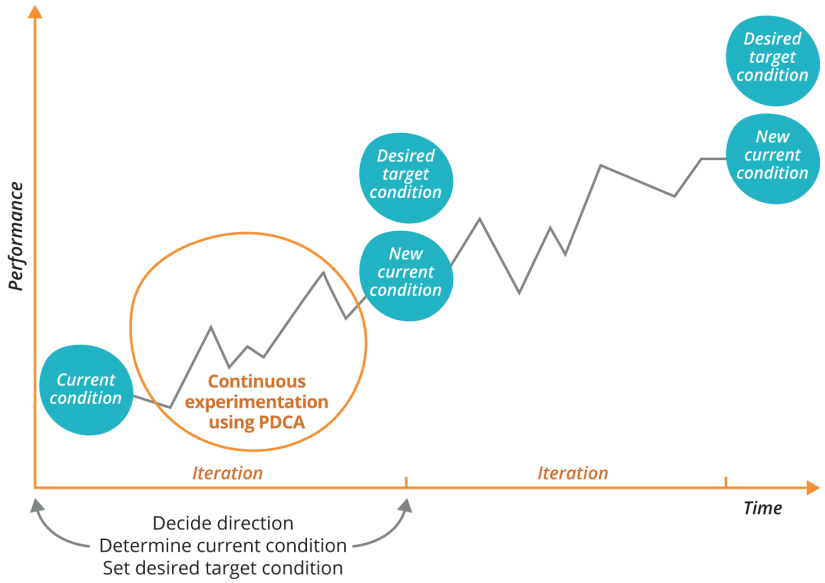


Figure 15-2. *Continuous evolution and adaption to change*

In order to propagate the Improvement Kata through organizations, managers must learn and deploy a complementary practice known as the Coaching Kata[1] To start the journey, an advance team including an executive sponsor—ideally, the CEO—should pilot the Coaching Kata and the Improvement Kata. As this team will guide wider adoption within the organization, it is imperative that they understand how it works.

Watch out for the following obstacles:

- Adopting the Improvement Kata requires substantial changes in behavior at all levels of the organization. The Coaching Kata is used to teach people the Improvement Kata, but the problem of how to deploy the Coaching Kata within an organization remains significant.
- Running experiments is hard and requires great discipline. Coming up with good experiments requires ingenuity and thought. By nature, people tend to jump straight to solutions instead of first agreeing on measurable target objectives (outcomes) and then working in rapid cycles—and by rapid, we mean hours or days—to create hypotheses, test, and learn from the results. The body of knowledge on how to design and run experiments

1 For free materials on the Improvement Kata and Coaching Kata, see *http://bit.ly/1v73SSg*.

in the context of product development is still in its early stages, and the necessary skills and techniques are not widely known or understood.

- Ensure there is capacity to run the Improvement Kata. One of the biggest obstacles teams face when trying to schedule improvement work is that it is often seen as a distraction from delivery work. This is a fallacy, and the point must be made early and forcefully. In the HP FutureSmart case, the reason delivery work was progressing so slowly was that no-value-add work was driving 95% of their costs. It is vital for executives at the director or VP level to ensure that teams limit their work in process as described in Chapter 7 to create time for improvement work.
- As with all methods, progress is likely to be bumpy at the beginning as people learn how to work in new ways. Things will get worse before they get better. Resistance is likely as people learn the new skills, and some will become frustrated when it conflicts with their existing habits and behaviors.

Aim Towards Strategy Deployment

Although we discussed the Improvement Kata as a way to drive continuous improvement at the program level, it can be used at every level from individual teams up to strategic planning. To apply the Improvement Kata at the strategic planning level, start by agreeing on the *purpose* of the organization. What is it that we aim to do for our customers? Then, those participating in the strategic planning exercise must define and agree upon the overall direction of the company—identify our "true north."

The next step is to understand and clarify our organization's current situation. Participants in the strategic planning exercise should identify which problems need to be addressed and gather data to better understand each problem. Typically, even large organizations have limited capacity and can manage only a handful of initiatives at any one time; choosing what *not* to focus on and making sure the team sticks to its decision is critical. An economic framework such as Cost of Delay (see Chapter 7) is useful to stimulate discussion about prioritizing work.

Once we have decided what problems to focus on, we need to define our target conditions. These target conditions should clearly communicate what success looks like; they must also include KPIs so we can measure our progress towards the goal. The traditional balanced scorecard approach to KPIs has four standard perspectives: finance, market, operations, and people and organization. Statoil, borrowing from the balanced scorecard approach in their Ambition to Action framework (Chapter 13), added HSE (health, safety, and environment). The lean movement teaches us to focus on reducing cost and

improving quality, delivery, morale, and safety (these five "lean metrics" are sometimes abbreviated as QCDMS). Bjarte Bogsnes, vice president of Performance Management Development at Statoil, recommends choosing 10–15 KPIs and preferring *relative* targets that connect input with outcomes (for example, unit cost rather than absolute cost) and are based on comparison with a baseline (for example, "10% higher return on capital investment than our leading competitor").[2]

The target objectives at the strategic level form the *direction* for the next organizational level, which then goes through its own Improvement Kata process. The target objectives at this level then form the direction for the next organizational level down, as shown in Figure 15-3. This process, allowing us to set targets and manage resources and performance by creating alignment between levels in the organization, is called *strategy deployment* (otherwise known as *Hoshin* or *Hoshin Kanri*; Ambition to Action is a variation of strategy deployment).[3]

The process of creating alignment and consensus between levels is critical. In strategy deployment, this process is described as *catchball*, a word chosen to evoke a collaborative exercise. The target conditions from one level should not be transcribed directly into the direction for teams working at the level below; catchball is more about *translation* of strategy, with "each layer interpreting and translating what objectives from the level above mean for it."[4] We should expect that feedback from teams will cause the higher-level plan to be updated. Don't subvert Hoshin by using it to simply cascade targets down through the organization: the key to Hoshin is that it is a mechanism for creating alignment based on collaboration and feedback loops at multiple levels.

The time horizons for each level should be clearly defined, and regular review meetings scheduled, with target objectives updated based on the progress of the next-level teams. To be truly effective, this conversation must also be cross-functional, promoting cooperation along value streams, within and between business units. It's not easy, as it requires honest listening to the ideas and concerns of the people responsible for results—and responding by adjusting the plans based on feedback.[5]

---

2 [bogsnes], pp. 125–126.

3 For a detailed description of Ambition to Action, see [bogsnes], pp. 114–169.

4 [bogsnes], p. 124.

5 Find out more about strategy deployment in Chapter 3 of Karen Martin's *The Outstanding Organization* [martin-12], and read a case study at *http://www.lean.org/Search/Documents/54.pdf*.

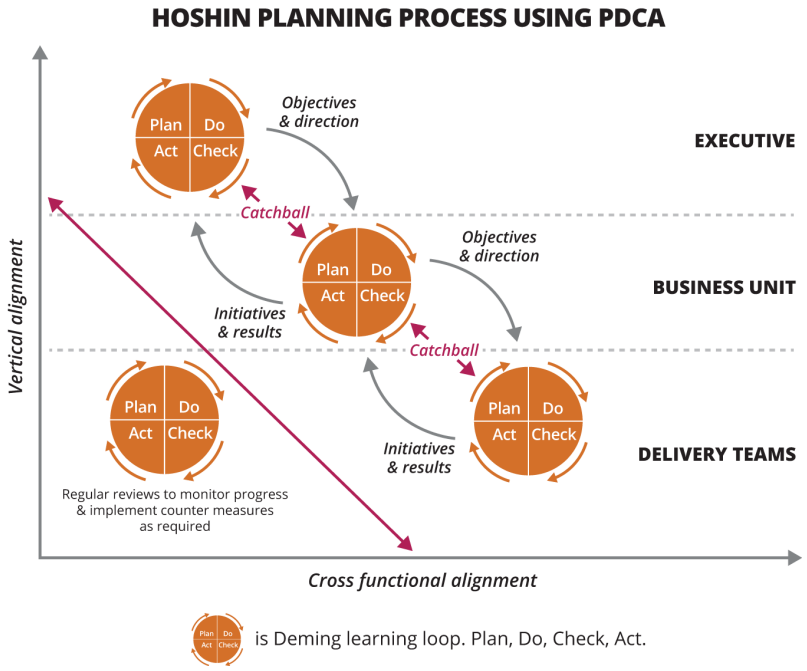


Figure 15-3. *Using catchball to drive strategic alignment of objectives and initiatives*

A top-level strategy planning exercise can have a horizon of six months to a few years, depending on what is appropriate to your business. Review meetings should be held at least monthly where the team, along with the leaders of all teams that report to them, gather to monitor progress and update target conditions in response to what they discover. Teams at lower levels will typically work to a shorter horizon, with more frequent review meetings.

Strategy deployment is an advanced tool that depends on the aligned culture and behaviors, as we describe in Chapter 11. The main goal is to create consensus and alignment and enable autonomy across the organization, following the Mission Command paradigm presented in Chapter 1. Let's look at how the UK government applied a version of strategy deployment to transform its use of digital platforms to provide services to citizens, starting small and growing iteratively and incrementally.

The UK Government Digital Service

The UK government, like many others, has recognized the potential of the Internet "both to communicate and interact better with citizens and to deliver significant efficiency savings."[6] However, government projects involving software development have a checkered past. The UK government had several large IT projects go enormously over budget while failing to deliver the expected benefits, culminating in the "National Programme for IT" debacle. The world's biggest civil information technology program, supposed to deliver a completely new IT infrastructure for the British National Health Service and a computerized patient record system, was projected to cost £2.3bn at its inception in 2002. Its delivery was outsourced to multiple private sector providers including Accenture, Computer Sciences Corporation, Fujitsu, and British Telecom. Despite the cancellation of the programme in 2011, it is expected to end up costing over £10bn.

The government procurement process for large IT projects involved writing a complete specification for the product, creating several business cases at increasing levels of detail, and then putting the contract out for bidding—a process that required one to two years before work could even start on the product. "By which time," comments Francis Maude, Minister for the UK's Cabinet Office, "it will almost certainly be out of date. You're locked into a supplier, it's really expensive to make changes."[7]

As a result of the outsourcing of IT projects, every government department had their own independently designed and operated web presence, with dissimilar user experiences that reflected each department's internal organization. It was complicated and extremely painful for citizens to use, so they preferred to use more expensive channels of service such as walk-in, mail, and phone services.

In 2010, Martha Lane Fox, co-founder of UK startup *lastminute.com*, was commissioned to advise the UK government on its strategy for online delivery of public services. Her report recommended creating a central team of civil servants responsible for designing and delivering the government's online presence, implementing an open data policy whereby all government data was made available through public APIs, and appointing a CEO "with absolute authority over the user experience across all government online services (websites and APIs) and the power to direct all government online spending."[8] Thus the UK's Government Digital Service (GDS) was born. Martha Lane Fox

6 [lane-fox]

7 *http://bit.ly/1F7yvbs*

8 [lane-fox]

described her goals for the GDS as follows: "For me, the acid test…is whether it can *empower, and make life simpler for, citizens* and at the same time allow government to *turn other things off*. A focus on vastly increasing the range, usage, and quality of online transactions will deliver the greatest impact: less hassle for citizens & businesses, and greater efficiency."

Government Digital Service Case Study, by Gareth Rushgrove

GOV.UK is the new single domain for all central government services in the UK. It was launched in October 2012 and replaced two of the largest existing government websites on day one, going on to replace all central government department sites over the next few months. By 2014 we will have closed thousands of websites and built a single service that is simpler, clearer, and faster, covering everything from information about benefits you may be eligible for to how to apply for a passport.

One aspect of GOV.UK that sets it apart from a typical government project is that it was developed nearly completely in-house, by civil servants working for the newly formed Government Digital Service (GDS), part of the UK Cabinet Office. It was also built iteratively, cheaply, and using agile methods and technologies more commonly associated with startups than large organizations. Here is a description of how that was done.

Alpha and Beta

By late 2013 the team running GOV.UK had over 100 people—but it didn't start that way. In fact, the first version wasn't even called GOV.UK. An Alpha version was built by 14 people working from a small back room in a large government building. Its aim wasn't to be a finished product but to provide a snapshot of what a single government website could be, and how it could be built quickly and cheaply. In total, the Alpha took 12 weeks and cost £261,000.

The feedback from users of the Alpha led directly to work on a Beta, which scaled up the Alpha proposition and involved more people from across government. The first release of the Beta was six months after the Alpha project shipped, but this included time to build up the team. The first public Beta release was a real government website, but at the time it lacked all the content and features needed to replace the existing main government sites. Eight months of constant iterations later, with the team up to 140 people and with new content and features added daily, traffic was redirected from two of the largest government websites to the new GOV.UK.

All this work paid off. During the financial year 2012–2013, GDS saved £42 million by replacing the Directgov and BusinessLink websites with GOV.UK. In 2013–2014, it is estimated GDS will save £50 million by closing more websites and bringing them onto the single domain.

Multidisciplinary Teams

The Government Digital Service is made up of specialists in software development, product management, design, user research, web operations, content design, and more, as well as specialists in government policy and other domain-specific areas. From this group of specialists, teams were formed to build and run GOV.UK. Those teams did not have a narrow focus, however; most of them were multidisciplinary, made up of people with the right mix of skills for the tasks at hand.

As an example, the team that worked on the initial stages of the Beta of GOV.UK consisted of seven developers, two designers, a product owner, two delivery managers, and five content designers. Even within these disciplines, a wide range of skills existed. The developers had skills ranging from frontend engineering to systems administration.

By employing multidisciplinary teams, the end-to-end responsibility for entire products or individual tasks could be pushed down to the team, removing the need for large-scale command and control. Such small self-contained teams had few dependencies on other teams so could move much more quickly.

This multidisciplinary model also helped to minimize problems typical in large organizations with siloed organizational structures. For instance, the Government Digital Service has grown over time, adding experts in government information assurance, procurement, and IT governance to avoid bottlenecks and improve the prioritization of resources.

Continuous Delivery

An important aspect of the success of GOV.UK has been constant improvement based on user feedback, testing, and web analytics data. The GOV.UK team releases new software on average about six times a day—with all kinds of improvements, from small bug fixes to completely new features, to the site and supporting platforms.

After the launch of the Beta of GOV.UK, one of the product managers, with bad memories of releasing software at other organizations, asked whether the software deployment mechanism was really going to work. The answer was "yes": at that point, GDS had done more than 1,000 deployments, so there was a high level of confidence. Practiced automation makes perfect.

This rate of releases is not typical for large organizations where existing processes sometimes appear designed to resist all change. The development teams working on GOV.UK worked extensively on automation, and they had in-depth conversations with people concerned about such rapid change. The key term when discussing this approach was *risk*—specifically, how regular releases can manage and minimize the risks of change.

Most people are bad at undertaking repetitive tasks, but computers are perfect for automating these tasks away. Deployment of software, especially if you are going to do it regularly, is a great candidate for automation. With the development and operation of GOV.UK, this was taken even further: provisioning of virtual machines, network configuration, firewall rules, and the infrastructure configuration were all automated. By describing large parts of the entire system in code, developers used tools like version control and unit testing to build trust in their changes, and focused on a smaller set of

well-practiced processes rather than a separate process (and requisite specialist skills) for each type of change.

Other techniques helped too. A relentless focus on users and a culture of trust from the very top of the organization have put GDS in a position to take much of what it learned building and running GOV.UK and use that to transform the rest of the UK government.

The GDS approach has been adopted by all arms of the government, with transformative results for citizens. To take just one example, the UK Ministry of Justice Digital Team recently worked with the National Offender Management Service and HM Prison Service to change the way people book prison visits. Previously, visitors had to request paper forms to be mailed out and then got on the phone to book a visit. Requests were often rejected because the date was unavailable, forcing people to start over. Now, prison visits can be booked online in 5 minutes, selecting from up to three dates.[9]

Not everybody is thrilled with the idea of governments growing their own IT capabilities. Tim Gregory, the UK president of CGI, the biggest contractor for the US HealthCare.gov website that received a contract valued at $292 million through 2013 before being replaced by Accenture in January 2014,[10] argues that the GDS approach will make it unprofitable for large outsourcing vendors to bid for government projects. GDS Executive Director Mike Bracken describes Gregory's view as "beyond parody."[11]

There are several observations to be made from the GDS case study.

First, starting small with a cross-functional team and gradually growing the capability of the product, while delivering value iteratively and incrementally, is an extremely effective way to mitigate the risks of replacing high-visibility systems, while simultaneously growing a high-performance culture. It provides a faster return on investment, substantial cost savings, and happier employees and users. This is possible even in a complex, highly regulated environment such as the government.

Second, instead of trying to replace existing systems and processes in a "big bang," the GDS replaced them incrementally, choosing to start where they could most quickly deliver value. They took the "strangler application" pattern presented in Chapter 10 and used it to effect both architectural *and* organizational change.

9 *http://bit.ly/1v73X8w*

10 Reuters: "As Obamacare tech woes mounted, contractor payments soared," *http://reut.rs/1v741oJ*.

11 *http://bit.ly/1v742ZT*

Third, the GDS pursued principle-based governance. The leadership team at GDS does not tell every person what to do but provides a set of guiding principles for people to make decisions aligned to the objectives of the organization. The GDS governance principles state:[12]

1. Don't slow down delivery.
2. Decide, when needed, at the right level.
3. Do it with the right people.
4. Go see for yourself.
5. Only do it if it adds value.
6. Trust and verify.

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.

Finally, the GDS shows that extraordinary levels of compensation and using a private sector model are not decisive for creating an innovation culture. GDS is staffed by civil servants, not Silicon Valley entrepreneurs with stock options.[13] 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.

Begin Your Journey

Use the following principles for getting started:[14]

Ensure you have a clearly defined direction
: The direction should succinctly express the business or describe organizational outcomes you wish to achieve in measurable terms, even if they look like an unachievable ideal. Most importantly, it should inspire everyone in the organization. Think of HP FutureSmart's goal of 10x productivity improvement.

12 GDS Governance principles, *http://bit.ly/1v747fT*.

13 In fact, the relatively low probability of a startup "exiting" successfully means that, for purely financial reasons, you'd be crazy to prefer a job at a startup over a solid position at (say) Google, as shown on slides 6–15 at *http://slidesha.re/1v6ZQZZ*.

14 These principles are partly inspired by John Kotter's eight-step process described in [kotter]: establish a sense of urgency, create the guiding coalition, develop a vision and strategy, communicate the change vision, empower broad-based action, generate short-term wins, consolidate gains and produce more change, anchor new approaches in the culture.

Define and limit your initial scope

Don't try to change the whole organization. Choose a small part of the organization—people who share your vision and have the capability to pursue it. As with the GDS, start with a single, cross-functional slice, perhaps a single product or service. Make sure you have support at all levels from executives down and from shop floor up. Create target objectives, but don't overthink them or plan how to achieve them. Ensure the team has what they need to experiment, follow the Improvement Kata, and iterate.

Pursue a high-performance culture of continuous improvement

Perhaps the most important outcome of deploying the Improvement Kata is to create an organization in which continuous improvement is a habit.

Start with the right people

New ways of working diffuse through organizations in the same way other innovations do, as we describe at the beginning of Chapter 2. The key is to find people who have a growth mindset (see Chapter 11) and are comfortable with trying out new ideas. Once you have achieved positive results, move on to the early adopters, followed by the early majority. The rest is relatively easy, because there's nothing the late majority hates more than being in the minority. This approach can be applied for each of the three horizons described in Chapter 2.

Find a way to deliver valuable, measurable results from early on

Although lasting change takes time and is never completed, it is essential to demonstrate real results quickly, as the GDS team did. Then, keep doing so to build momentum and credibility. In fact, the Improvement Kata strategy is designed to achieve this goal, which we hope will make it attractive to executives who typically have to demonstrate results quickly and consistently on a tight budget.

As you experiment and learn, share what works and what doesn't. Run regular showcases inviting key stakeholders in the organization and your next adoption segment. Hold retrospectives to reflect on what you have achieved and use them to update and refine your vision. Always, keep moving forward. Fear, uncertainty, and discomfort are your compasses toward growth. You can start right now by filling out the simple one-page form shown in Figure 15-4 (see Chapter 11 for more details on target conditions). For more on how to create sustainable change, particularly in the absence of executive support, we recommend *Fearless Change: Patterns for Introducing New Ideas* by Mary Lynn Manns and Linda Rising.[15]

15 [manns]

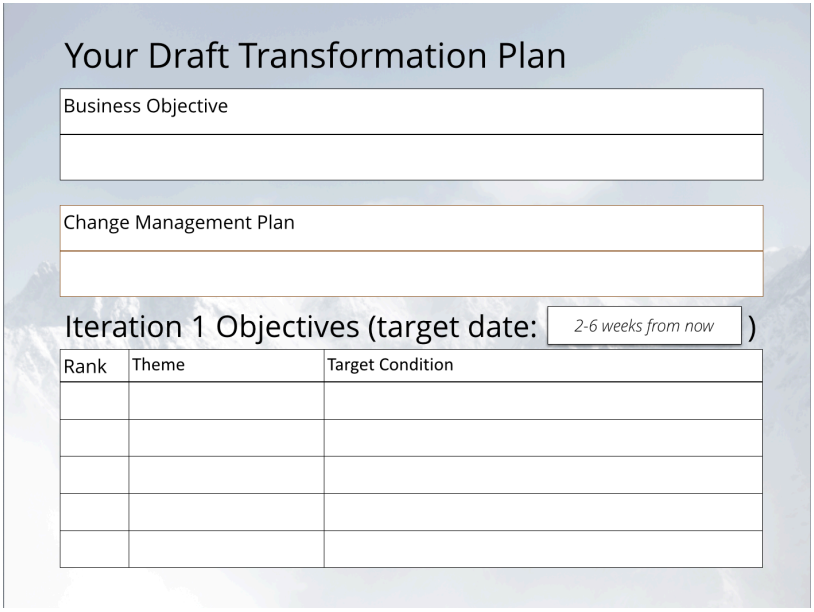


Figure 15-4. *Draft transformation plan*

Conclusion

Creating a resilient, lean enterprise that can adapt rapidly to changing conditions relies on a culture of learning through experimentation. For this culture to thrive, the whole organization must be aware of its purpose and work continuously to understand the current conditions, set short-term target conditions, and enable people to experiment to achieve them. We then reassess our current conditions, update our target conditions based on what we learned, and keep going. This behavior must become habitual and pervasive. That is how we create a mindset of continuous improvement focused on ever higher levels of customer service and quality at ever lower costs.

These principles are the threads that link all scientific patterns together. Whether you're seeking a repeatable business model through the Lean Startup learning loop, working to improve your product through user research and continuous delivery, or driving process innovation and organizational change using PDCA cycles of the Improvement Kata—all that is based on a disciplined, rigorous pursuit of innovation in conditions of uncertainty. That the same principles are at the heart of both lean product development and effective process and cultural change was an epiphany for the authors of this book, but perhaps it should not be a surprise—in both cases we face uncertainty and have to deal with a complex adaptive system whose response to change is

unpredictable. Both these situations call for iterative, incremental progress, achieved through human creativity harnessed by the scientific method.

Organizations must continually revisit the question: "What is our purpose, and how can we organize to increase our long-term potential and that of our customers and employees?" The most important work for leaders is to pursue the high-performance culture described in this book. In this way, we can prosper in an environment of constant advances in design and technology and wider social and economic change.