Part I
DevOps: Conflict to Collaboration

1. DevOps in the Ascendency

Aruna Ravichandran[1]✉, Kieran Taylor[2] and Peter Waterhouse[3]

(1)   CA Technologies, Cupertino, California, USA
(2)   Winchester, Massachusetts, USA
(3)   Blackburn, Victoria, Australia

"Be ahead of the times through endless creativity, inquisitiveness, and pursuit of improvement."
          —The Toyota Precepts[1]

Accelerating Agile Practices in Today's Software Factory

In 2016, Formula 1 (FI) racecars, the ultimate in four-wheeled technology, are awash in wireless sensors and transmitters.

Running your eyes across the graceful, sculpted bodywork of these 200+ mph engineering wonders, the only discernible interruption of their low-slung carbon fiber noses are small clusters of wireless antennas jutting up directly into the drivers' sightlines.

As subtle as these transmitters may be, why would designers, who spend endless time and resources attempting to improve the aerodynamics and drivability of the machines, accept such a stark concession? The answer is simple—to arm their teams with priceless real-time data used to continuously improve performance.

Along the pit row wall, in the trackside garages, and back in the very design centers where F1 engineers continue to refine their handiwork, telemetric information provided by those tiny antennae is immediately translated into

change.

While team principals and technical directors sitting directly adjacent to the track communicate directly with the drivers, advising them how to adjust settings on the fly, their colleagues in the garages prepare to adjust everything from tires to aerodynamics when the machines pull into the pits.

Meanwhile, back at the factory, live data is consumed by all manner of engineering teams who continually produce and modify new components to be utilized in subsequent competitions—beginning the process of further advancement even before the current race is complete. The push for innovation in F1 is a perpetual activity; in this sense, perhaps more so than in any other sport, the only constant is technological change.

That Toyota Motor Corp., the global automotive giant best known for advancing such data-driven, process-centric "just in time" manufacturing practices—concepts that revolutionized mass production of passenger cars—had such a short-lived and disappointing record in F1 can only be classified as ironic. However, the Kanban and Kata methodologies leveraged by Toyota since its inception, paralleled on a smaller, more specialized scale by today's F1 teams, stand as the most widely emulated management frameworks in history.

With an emphasis on the adoption and evolution of processes designed to optimize resources while increasing production speed and quality—and most importantly promoting constant innovation—the "Toyota Way"[2] represents arguably the leading model for continuous improvement.

In the words of W. Edwards Deming—the legendary American engineer and management consultant who helped spearhead the reinvention of Japanese industry after WW2—Toyota's approach embodies the notion that, "It is not enough to do your best; you must know what to do, and then do your best."[3]

Grounded in constant observation and measurement of efficiency—from supply chain management through final production, with the goal of constantly improving output, including and in particular worker productivity—this methodology flies in the face of traditional "waterfall" workflows. American automaker Henry Ford may be credited with revolutionizing the assembly line, but Toyota is widely recognized for decoupling and perfecting it.

Rather than creating linear dependencies, where each successive activity is wholly dependent on completion of the task preceding it, this revolutionary approach emphasizes techniques wherein production is dynamically adapted on the fly to result in optimal efficiency and maximum quality.

Further illustrating the staunch devotion to continuous improvement represented both in the Toyota Way and in his personal doctrine, Deming

famously said, "It is not necessary to change. Survival is not mandatory."[4]

This observation is neither subtle, nor, given industrial history, seemingly misplaced.

Embracing DevOps in the Application Economy

In today's rapidly evolving Application Economy, it is widely recognized that, driven by the evolution of digital channels—across both the business-to-business and consumer segments—many organizations are reinventing or recasting themselves as providers of software and digital services.

For example, the global population of mobile banking users is forecast to double to 1.8 billion by 2020, representing over 25 percent of the world's population, according to KPMG.[5] As a result, banks are increasingly focused on advancement of web-based and mobile applications, versus expansion of brick-and-mortar operations.

Furthermore, as business delivery mechanisms shift to the digital landscape, end users' tolerance of latency of such applications has grown increasingly narrow. According to a study published by **Wired** in June 2014, roughly 50 percent of consumers expect a web page to load in two seconds or less—or they move to abandon it.[6]

Given this transformation, as traditional business services are replaced by largely web-based and mobile-friendly applications, organizations are being forced to completely reexamine their software development and IT management practices. Technology is no longer viewed as a supporting dependency, but rather as a primary element of conducting business.

Within that context, most of today's organizations are moving aggressively to adopt more agile, efficient software delivery and IT management practices to meet customers' evolving expectations. Among the fastest growing and most widely adopted strategies, aimed at bringing Toyota-like efficiency to the world of the applications lifecycle, is the methodology known as **DevOps**.

Whether or not Patrick Debois understood that he was creating, or at the very least putting a name to, the movement that would completely redefine the manner in which organizations build and support software is unclear. What is known is that since Debois, a systems administrator, and a handful of like-minded software development and IT operations experts first coined the DevOps moniker in 2009, the concept has become a global phenomenon.

The underlying concepts encompassed by DevOps are, at first glance, straightforward, but represent seismic reformulation within the context of

software production and support. Rather than maintaining discreet applications engineering ("Dev") and IT management ("Ops") competencies and organizations, DevOps dictates use of smaller teams with cross-functional expertise to improve software functionality and the processes used to deliver it.

As highlighted in the seminal DevOps novel, **The Phoenix Project**, this mindset eliminates the fragmented approach to applications delivery that has traditionally crippled many organizations in addressing the digital market opportunity. Rather than asking developers to build an application and then charging IT management with ongoing support, creating highly disparate and inefficient dynamics, DevOps brings those specialists together so that engineering is completed with a constant eye toward ongoing management.

Just as critical in boosting organizational efficiency and software quality, DevOps methodology—like the Toyota Way—also promises to increase the ability to change the existing code base and deliver new capabilities to end users, while cultivating internal experimentation. Leveraging automation to do so is another hallmark of the Japanese carmaker's model and is also core to the DevOps approach.

In a May 2014 editorial published in the **Wall Street Journal,** noted technology evangelist and **The Phoenix Project** co-author Gene Kim echoed the words of Deming in framing his view of ongoing DevOps adoption. Titled "Enterprise DevOps Adoption Isn't Mandatory — but Neither Is Survival," Kim, also an established entrepreneur, posits that "the business value created by DevOps will be even larger than was created by the manufacturing revolution."[7]

Responding to some experts' observations that DevOps is only relevant for lean-minded, applications-driven startups such as consumer transportation darling Uber, Kim asserts that even the oldest, most entrenched businesses must wrap their arms around the movement to compete and survive. The author contends in the WSJ piece that this is, "because IT is the factory floor of this century, and not just for manufacturing companies. IT is increasingly how all businesses acquire customers and deliver value to them."

Like the so-called "lean manufacturing" wave that swept the business world in the 1980s—directly related to the success of Toyota in growing to the point of global sales domination—Kim asserts that today's organizations must seek to drive every element of inefficiency out of their software development lifecycle (SDLC).

Just as companies that failed to adopt leaner manufacturing processes in the latter half of the 20th Century lost out to rivals that did, the expert, among many others, maintains that organizations who overlook the need to adopt DevOps in today's Applications Economy will likely disappear.

Evidence that Kim and other proponents are correct in predicting that organizations' willingness to embrace DevOps will directly impact their ability to compete is already mounting. Numerous research reports have reinforced that leading DevOps practitioners already appreciate significant competitive benefits.

In a 2015 study published by Freeform Dynamics, in partnership with CA Technologies, researchers found that the 20 percent of organizations that had broadly adopted DevOps methodologies were 2.5 times more likely to have charted improvements in customer retention.

The report, "Assembling the DevOps Jigsaw,"[8] also found that these early adopters were 2 times more likely to have realized improvements in customer acquisition, and 3.4 times more likely to appreciate growth in market share. Perhaps most notably, those organizations already well into their DevOps transformations were 2 times more likely to have recorded a positive impact on revenue, and 2.4 times more likely to have experienced higher profit growth.

Regardless of the given era and environment, those financial results would seem to speak for themselves. Yet, according to the Freeform Dynamics survey, 80 percent of all organizations have yet to embrace DevOps completely.

DevOps as a Critical Requirement

The reality is, whether organizations are ready or not, the requirement to aggressively adopt DevOps methodology has quickly become the new normal in the Applications Economy. At the very least, the notion that such change is inevitable in cultivating continued business growth must be recognized as an inconvenient, yet irrefutable truth.

In an increasingly fast-paced, complex, and ambiguous business world, competing on a landscape dominated by the continued evolution of digital channels and mobile devices, the only tractable strategy is to adapt to survive. Driven by the exponential speed of change, organizations must wrap their arms around DevOps if they hope to defend and expand their market opportunities.

Within this current atmosphere—defined by volatility, uncertainty, complexity, and ambiguity (VUCA)—DevOps offers an unprecedented opportunity for organizations to transform their SDLC to increase efficiency and meet end users' changing expectations. By fundamentally recasting the manner in which they approach every element of software development and management, DevOps represents the broader future of business in general.

Despite the tendency to celebrate industry "unicorns" such as Uber—startups whose technology and business models immediately lent themselves to DevOps

adoption—research such as the Freeform Dynamics "Jigsaw" report illustrates that organizations of all sizes and industries must change, or risk potential obsolescence. This conclusion is already being proven out by widespread assimilation of the involved methodologies by everyone from lean startups and centuries-old manufacturers, to nonprofits and government agencies.

Banking on DevOps Practices

Adoption among stalwart names in the banking industry—recognized as one of the most entrenched and "old school" segments in the entire business world—offers further proof of this pervasive need for DevOps transformation.

While notoriously staid in some senses, banks have also long-embraced significant levels of software and IT to diversify their services and increase profitability. Examples include inventions such as automated teller machines (ATMs) and the vast electronic transaction processing systems that allow these companies and their clients to move capital around the world in real time.

With roots dating back over 150+ years into the Dutch banking trade, global corporation ING is one such old world company that has successfully embraced DevOps transformation. Driven by existing inefficiencies in its 15,000-strong IT workforce and the need to address changing customer demands to increase its (EU) 15 billion annual revenues, the Amsterdam-based company embarked on an aggressive DevOps strategy beginning in 2011.[9]

Aimed specifically at enabling so-called "continuous delivery" of its applications to suit emerging customer preferences around online and mobile banking, company officials sought to remodel SDLC methodologies by invoking a manifesto of legendary Apple co-founder Steve Jobs. Citing a 1989 interview with *Inc. Magazine* in which Jobs famously said, "You can't just ask customers what they want and then try to give that to them. By the time you get it built, they'll want something new,"[10] ING set about on its DevOps initiative.

With the expressed goal of retrenching its SDLC to eliminate onerous, time-consuming waterfall practices and accelerate the pace of applications innovation, ING first moved to create more agile "scrum" teams that brought together development and operations expertise. From a process standpoint, one of ING's tactical goals was to begin testing new applications code in a far more "production-like" environment, such that emerging issues could be identified and resolved faster, accelerating code delivery to end users, while improving quality.

In support of those initiatives, the banking giant also leveraged significant

automation to address SDLC requirements, including configuration management and software deployment. By bridging the gap between dev and ops, and employing new levels of automation, the company was able to increase the pace of its applications releases from an average of once every 13 weeks to weekly updates.

In support of this larger continuous delivery effort, ING was also able to increase its pace of underlying mobile applications code deployments from several hundred per month to over 80,000 per month, in less than two years. By leveraging DevOps workflows and supporting automation, the Dutch bank transitioned from release cycles dictated by technical roadmaps, to those focused on strategic business objectives.

Ultimately, customers expressed their approval both in adoption and via public forums, as ING's mobile application reviews on the Apple iTunes App Store climbed from one star in 2011, to four stars in 2014.

DevOps: A Key Component of Business Agility

The ING story stands out not only as detailed proof of the manner in which DevOps represents a tremendous opportunity SDLC reinvention, but also as a specific example of how large, entrenched organizations can also emulate the much admired startup mentality. At the end of the day, the company's continuous delivery exercise allowed it to appease the changing preferences of mobile applications users, the ultimate goal of countless new startup ventures.

For years, everyone from management consultants to Wall Street investors have beleaguered the need for such pre-Internet companies to better leverage the "lean and mean" traits of their upstart peers. With the dawn of the DevOps era, driven by the Applications Economy, organizations, regardless of scale and history, have in fact been presented with this specific opportunity.

By completely transforming the manner, and more importantly the speed with which new ideas can be translated into marketable products and services—in the specific form of applications—organizations that have traditionally been slowed by their inherent size and complexity can rapidly accelerate innovation.

Conventional wisdom dictates that large enterprises with sundry customers and products face greater risk in attempting to launch new services and supporting business models. Meanwhile startups, unburdened with existing processes and systems, can rapidly pivot from one guiding approach to another, reinventing themselves whenever the need arises. Enterprises, typically guided by the need to requite shareholder interests, have been prone to err more toward stability, limiting their rapid growth potential.

For example, it is widely recognized that this entrenched, risk-averse mentality has allowed startup companies such as eBay and Airbnb to displace longstanding giants of the global retail and hospitality industries, respectively. If you could turn back the clock 5-10 years and give Sears and Hilton another chance to adopt the same strategies used by their startup rivals, obviously they would jump at the opportunity.

Part of this, in addition to a lack of shareholder reckoning, revolves around the fact that startups have also been able to quickly embrace emerging business technology trends, including cloud computing, mobile apps, open source, crowdfunding, and other means of collaborative economics.

Yet, if the core of this mentality, empowered by technological agility, is that innovation is merely a "good idea made possible," then adoption of DevOps represents just such an opportunity for organizations, regardless of size, to accelerate business via more efficient applications delivery.

As framed by Jez Humble, leading technology consultant, recognized as a founding father of the DevOps movement: "The long-term value of an enterprise is not captured by the value of its products and intellectual property, but rather by its ability to continuously increase the value it provides to customers—and to create new customers—through innovation."[11]

At its core, the premium placed on the startup mentality, as previously referenced, relates primarily to the notion of market disruption. This is perhaps best exemplified, thus far, by those Applications Economy darlings that have successfully introduced new business models supported by web and mobile technologies.

However, leveraging DevOps and its halo effect of increased innovation through more efficient and rapid delivery of differentiated applications, one could easily assert that in the matter of enterprise versus startup, the playing field is being leveled quickly. Specifically, as organizations leverage DevOps to become more agile in addressing the Applications Economy, they are able to better keep pace with their smaller, more innately nimble rivals.

Circling back to the origins of the Toyota Way, the involved tenets of disruption via technological innovation have always been recognized as catalyst in transforming industry; in fact, even the genesis of this specific example has roots predating the auto industry.

In 1896, Sakichi Toyoda invented Japan's first self-powered loom, incorporating numerous revolutionary features, most notably the ability to automatically halt production when threads moving through the devices were broken.[12] Leveraging his invention, Toyoda built his startup into a leader in the

Japanese textiles industry.

Those concepts, which became the legendary Toyota Way, were merely carried over when his son Kiichiro launched the automaker Toyota Motors in 1937. As it would turn out, this technology-driven approach to innovation and disruption would also dovetail perfectly with the lean management concepts promoted by Deming as he helped resurrect Japanese manufacturing in the 1940s.

At the time, automakers in other areas of the globe, notably the United States, likely would have never accepted that such a company, far smaller than their own booming operations and literally rising from the ashes of war, would have the opportunity to dominate the worldwide market.

In 2012, after decades of displacing customers from other brands, Toyota was recognized at the world's largest automobile manufacturer, having produced its 200-millionth vehicle and grown into the first automobile manufacturer to ever produce more than 10 million vehicles per year. In 2016, amid larger worldwide economic concerns and a headline-grabbing airbag recall issue, the company remains among the global sales leaders in its industry.

Companies such as Toyota and ING, along with endless agile startups, offer tacit proof that for organizations seeking to increase market relevance and disruption, DevOps offers an attractive template. Yet, citing Freeform Dynamics' findings that only 20 percent of all organizations have achieved any semblance of DevOps maturity, the movement is clearly still in its nascence.

To wit, the researchers found that even though 80 percent of respondents to its survey view DevOps as a "key component of business agility," only 55 percent of organizations laid claim to having created a well-defined DevOps strategy. Furthermore, while 86 percent of those organizations surveyed labeled DevOps-oriented education of business stakeholders and greater alignment of IT and business priorities as important, only 33 percent and 37 percent, had enacted those programs, respectively.

Loosely stated, the concept of disruption, in general, suggests that market incumbents typically become less profitable and open to displacement from more agile peers based on the entrenchment of inflexible business models. It is widely acknowledged that there are two primary models for market disruption in the current era—"new market" disruption and "low end" disruption—at least as defined within the context of so-called "disruptive innovation" theory.[13]

In the case of new market disruption, the landscape is forever altered when a new products arrives that somehow better addresses a segment that incumbents have not yet envisioned or delivered. Whereas with low end disruption, incumbents see business chipped away by products that aim to encompass a

smaller subset of perceived value drivers offered at a lower price point.

Looking closer at the promise and outfalls of DevOps—and the ability of adopters to better align themselves with the exploding demand for software and applications—one could easily argue that this ongoing technological revolution directly supports both archetypes of the disruptive opportunity. It would seem obvious that those organizations that have yet to jump in sit greatly imperiled by its continued advancement.

Some may view the Darwinian views of Deming, or Gene Kim, as overstated, particularly as related to DevOps and the Applications Economy. One could imagine that organizations that do not yet supply applications to consumers or business partners would remove themselves from the larger conversation.

However, in the following chapters of this book, there is a great deal of evidence—backed by specific technical practices—that make it clear just how sensible, if not necessary, it is for every organization to view the DevOps opportunity as their best chance for success and survival.

DevOps: A Practice for Champions

At the conclusion of the 2015 F1 season, the world champion driver Lewis Hamilton and teammate Nico Rosberg of the Mercedes AMG Petronas Formula 1 Team, also winners of the highly coveted F1 constructor's title, returned to the outfit's UK headquarters to thank the factory workers who made their victories possible.

While most of those people, numbering in the hundreds, never joined the team at the track on a single race day during the globe-spanning nine month season, the champions heaped praise on their colleagues, recognizing the year-round effort that supports the entire operation.

Mercedes made easy work of the rest of the F1 field in 2105, having won a stunning 16 of the 19 total races. At the same time, this was not without constant updates to its racecars, with new components constantly meeting the team around the world as they progressed throughout the grueling race calendar.

"In racing there are always things you can learn, every single day. There is always space for improvement, and I think that applies to everything in life," Hamilton has been quoted as saying.[14]

"With the team, whether it be my guys that are at each Grand Prix, the team back at the factory who work day and night to develop the car, build the parts, and send the parts, the IT team, and the crew cleaning the factory. Everyone. We

do it together. We all feel the joy when we are winning and the same pain when we are losing," the champion driver continued.

Without the constant delivery of new innovations, without the continued experimentation and refinement of the involved technologies, and of course all the underlying processes, Hamilton affirms, the likelihood of his success would be scant, if not impossible.

DevOps offers the chance for any organization to run more like a championship team, ever improving performance, going faster, and winning.

Summary

Only a few years into the Application Economy it's clear that huge benefits are being appreciated by those organizations capable of embracing emerging processes and technologies that enable agile methodologies. DevOps is widely recognized as one of the most influential and important movements that empower that transformation.

At the same time, such a sea change in the manner that organizations approach software development and operation, or any process for that manner, cannot be realized without some growing pains. In the next chapter, we'll take a look at the historic state of disconnect that has existed between developers and IT operations staff, and the approach that organizations embracing DevOps have employed to reinvent the manner that such experts collaborate.

Footnotes

1 "Toyota Quotes," *The New York Times,* Feb. 10, 2008:
   http://www.nytimes.com/2008/02/10/business/worldbusiness/10iht-10facts.9900222.html?_r=0

2 Toyota Motor Corporation Annual Report, 2003, page 19.

3 "Made in Japan," `MadeinWyoming.com`, Rena Delbridge, copyright 1995:
   `http://madeinwyoming.net/profiles/deming.php`

4 Edward Deming, *Out of the Crisis,* (Cambridge, MA, MIT Press, 1982), p. 227.

5 "Global Mobile Banking Report," KPMG LLP, copyright 2015:
   `http://www.kpmg.com/channelislands/en/IssuesAndInsights/ArticlesPublica`
   `%20May%202015.pdf`

6 "Great Expectations: 47% of Consumers Want a Web Page to Load in Two Seconds or Less," by Nilesh
   Patel, *Wired*, copyright 2014: `http://insights.wired.com/profiles/blogs/47-of-`
   `consumers-expect-a-web-page-to-load-in-2-seconds-or-less#ixzz40vkvFwVl`

7 "Enterprise DevOps Adoption Isn't Mandatory — but Neither Is Survival," by Gene Kim, copyright
   WSJ, 2014:
   `http://blogs.wsj.com/cio/2014/05/22/enterprise-devops-adoption-isnt-`
   `mandatory-but-neither-is-survival/`

8 "Assembling the DevOps Jigsaw," Freeform Dynamics, copyright 2015:
   `http://rewrite.ca.com/us/articles/devops/assembling-the-devops-`
   `jigsaw.html`

9 "ING Bank Case Study," CA World'14 Presentation copyright CA Technologies.
   `https://www.youtube.com/watch?v=9jqY_bvI5vk`

10 "The Entrepreneur of the Decade," by George Gendron and Bo Burlingham, copyright *Inc. Magazine*
   1989: `http://fortune.com/2009/11/23/decade-steve-jobs-apple/`

11 Jez Humble, *Lean Enterprise: How High Performance Organizations Innovate at Scale,* (Sebastopol, California, O'Reilly, 2015) p. 39.

12 "Automation with a Human Touch," Toyota Motor Corporation World site, copyright 2016: http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/jidoka.ht

13 "What is Disruptive Innovation?," copyright Harvard Business Review, Dec. 2015.

14 "Mercedes AMG Petronas Team End Season on Top," By Donny Halliwell, *Inside Blackberry*, copyright 2015. http://crackberry.com/mercedes-amg-petronas-formula-one-team

2. IT Impasse

Faster Software Development Versus Operational Stability

Aruna Ravichandran[1]✉, Kieran Taylor[2] and
Peter Waterhouse[3]

(1) CA Technologies, Cupertino, California, USA
(2) Winchester, Massachusetts, USA
(3) Blackburn, Victoria, Australia

Ever since we flipped the switch on commercial computers back in the 1950s, IT departments have been struggling to keep up with an insatiable demand for software applications and services. Of course many technologies like commercial of-the-shelf software packages, virtualization, and cloud computing have helped along the way, but generally IT delivery has been slow and uncoordinated.

In such an environment where IT generally supported internally-centric business processes, separate teams overseeing discrete technology functions was for many years considered the norm. But in today's rapidly evolving digital world, these practices no longer work.

A World of 'Wicked' Business Problems

Keeping up with demand for changes to internal applications supporting business processes is tame compared to the "wicked" problems facing IT departments today. Like societal problems such as global warming and drug abuse, they're wicked because IT is placed in an unenviable position of trying to

rapidly deliver solutions before problems are fully understood and where business conditions constantly change. This is due to three transformational forces:

- ***Products to services***—Customers now care less about physical things and more about the total experience. This explains why companies wrap physical products in services. Like Tesla, who routinely deliver enhancements to the Model S car via software. Or Bosch, who now provides tailored telematics and analytics as-a-services so that fleet operators can optimize maintenance and cut fuel costs.

- ***Efficiency to agility***—Established brands with decades of reputation are constantly being disrupted. Kodak lasted 100 years, Blockbuster less than 20. Even technology stalwarts like Microsoft have felt the ground shift beneath them. Business reputations are forged from digital adeptness and the ability stay ahead of the market—or create new ones.

- ***Separation to fusion***—There is no longer separation between physical products and software. Is your smartphone circuitry and plastic, or is it a Spotify music service and digital payment system? Is your Nest home thermostat an aesthetically pleasing appliance, or is it an analytical marvel of energy management?

Operating within this environment, wicked problems now challenge the essence of how business is conducted and how applications should be designed, developed, and supported. Thanks to mobility and social computing, the producer-consumer relationship has been reversed, with businesses placed in a position of having to respond to customer behaviors rather than dictating them. This places many organizations on the back foot because of traditional development practices.

Internal business processes supported by large complex applications have been the lifeblood of business. Supporting processes like logistics, inventory, and financial management, these applications are periodically optimized to eke out operational cost efficiencies. In this context, the focus on IT has been to maintain a steady state; keeping the technical lights and only changing applications over longer cycles, sometimes only once or twice a year.

Even when custom development occurs, the general practice has been to invest heavily in application software to support large-scale projects. Here, the pattern is to define all the requirements and features at the start and progress the application toward production status by developing and testing in a linear fashion. Finally, if many business stakeholders agree the system is what's

wanted, the application is handed over the wall to production for IT operations to maintain.

Considering these forces, this "Waterfall" style of development (see Figure 2-1) now has a number of disadvantages. First, by establishing a fully defined set of requirements up-front and then failing to accommodate shifting customer behaviors, organizations risk delivering "white elephant" applications, which are software systems that customers never use. Secondly, in the time taken to release software, business conditions may have changed, meaning departments must change the system radically, or as is often the case, abandon it completely. Finally, since application software and functionality is delivered en masse, the support and maintenance burden on IT operations increases suddenly and significantly.
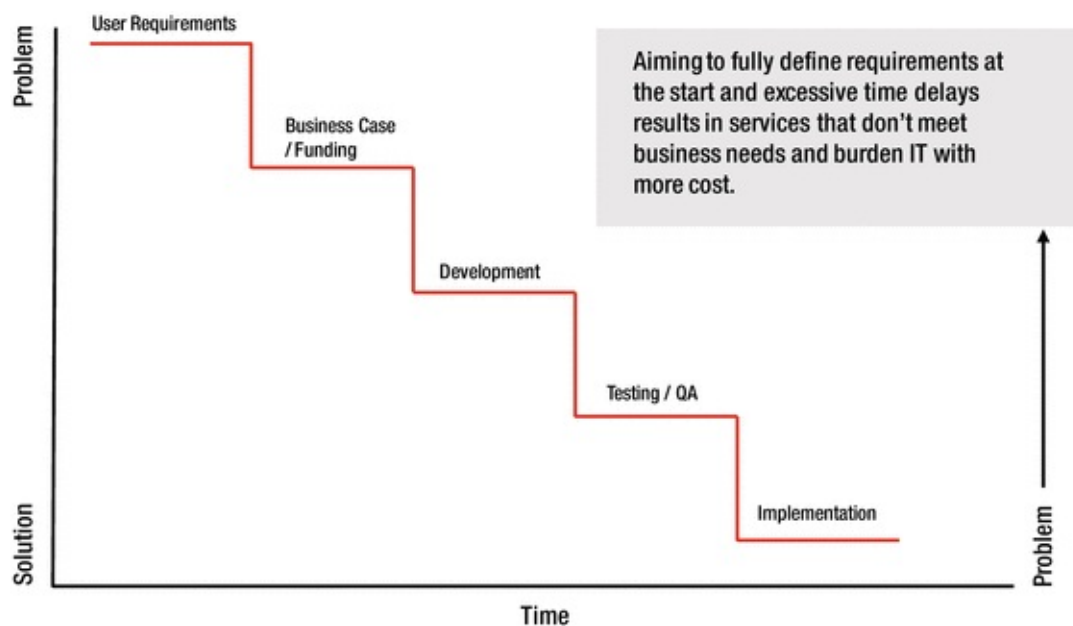

Figure 2-1. Waterfall software development model

The Emergence of Agile Development

In Formula 1 racing, rules change every season. Cars now compete without fuel pit-stops and on sets of tires that degrade more quickly. It's the same in IT, where software must now be delivered as a continuous stream of value that constantly changes to support new business conditions.

Trying to solve wicked business problems (e.g., transforming the businesses operational model with a new mobile sales channel) with existing methodologies like Waterfall can be like trying to corral cats. Problem definitions and

requirements can change as new solutions are considered and implemented, so much more flexibility is required across the software development lifecycle. And, with many diverse opinions on what actually constitutes a business problem, there'll be many stakeholders to engage and ideas to analyze.

Agile development has emerged as the practice of choice to address these challenges. Agile (see Figure 2-2), which involves iterative problem solving and continuous feedback, is well suited to solving these complex business problems. Some common benefits include:

- Software updates are made in small regular batches, allowing businesses to adapt quickly to new information

- Mistakes or false assumptions are detected much earlier in the software lifecycle, where they are much less costly to correct

- By detecting problems quickly and early, teams have much faster feedback and organizational knowledge improves

- Teams can immediately apply learning and knowledge gained to improve the quality of application software

- By working in parallel across development and testing, agile teams can increase the velocity of application development and delivery
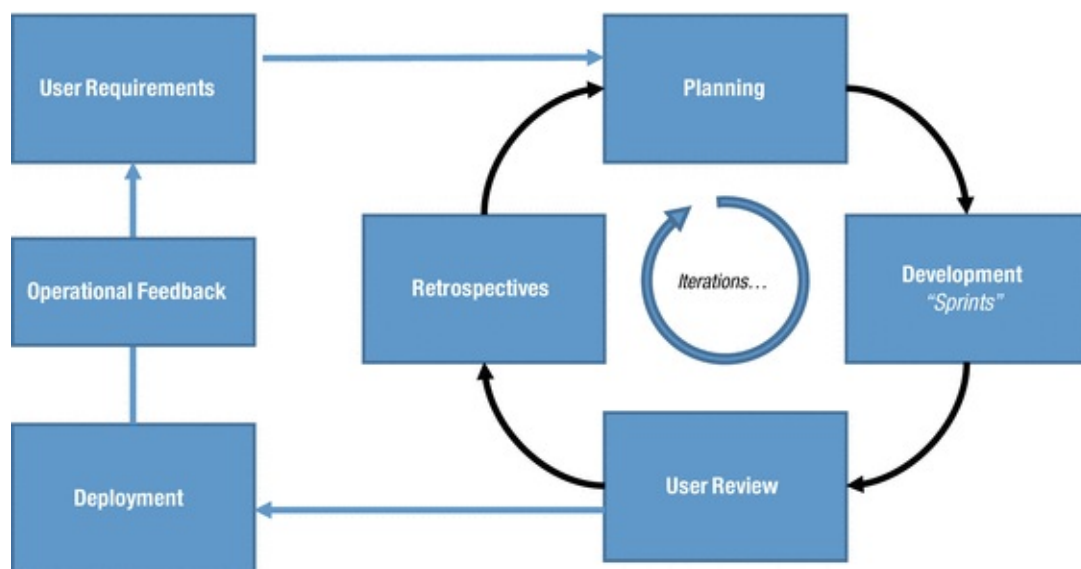

Figure 2-2. Agile software development model

Agile methods help teams manage and run their own projects. Rather than wait for approvals and sign-offs across multiple stages, agile supports the notion

that fully autonomous teams should be able to make their own architectural design decisions, even when it comes to tool usage and deciding what's needed to push software code all the way into production.

But agile development is not without its drawbacks. While agile development teams are focused on speed and agility, the traditional mantra of IT operations is maintaining application stability, even if that means slowing things down. To this end, IT operations often employ rigorous change control processes and enforce infrastructure standardization dictates to maintain control. While well suited to supporting legacy internal systems where updates were delivered in large batches, these processes can be too rigid to support newer applications designed to be always changing.

Many of these issues are a direct consequence of how IT operations have traditionally been run. Historically, IT operations have been delivered as a set of shared services, supported by functional disciplines in areas such as data center services, network management, and security. If business users or development needed an enabling service (e.g., provisioning test labs), operations had to be engaged for delivery.

However, this model is changing, with new tools and methods that enable business groups and development teams to bypass the IT operations shared service completely. Many recent technology trends have supported these practices, including:

- **Polyglot programming languages/platforms**—Autonomous teams can select the programming language best suited to their project. This may include older languages like Java and C++, or new platforms and languages to support specific development use-cases or design requirements (e.g., Node.JS, Go, and Rust). In other cases development teams are eschewing traditional relational databases in favor of NoSQL or document style data stores. These could be advantageous in projects where data requirements are indeterminate or evolving and where speed and scalability is more critical than up-front logical design and data integrity. Examples include MongoDB, PostgreSQL, and Cassandra.

- **Programmable Infrastructure**—Also known as Infrastructure-as-code, this allows teams to write code to manage configurations and automate infrastructure provisioning. Rather than using manual methods or scripting to build development ready infrastructure, teams can write code in languages with which they are familiar, together with familiar practices such as version control and automated testing to reduce errors. Unlike scripting, which is primarily used to automate static steps, programmable

infrastructure with its descriptive languages allows developers to code processes that are far more versatile and adaptive. Examples include Chef, Puppet, and Ansible.

- *Open source software*—Beyond the more obvious "free to use" benefits, many organizations are embracing open source software (OSS) as a means to increase agility. By adopting OSS, development teams can benefit from community-based and collaborative development, with tools that simplify and automate many tasks (e.g., version control and software build management). By having the eyes of the "community" on the software, bugs can be quickly identified and resolved, with new improvements constantly being developed and delivered.

- *Cloud and platform-as-a-service (PaaS)*—Not too long ago testing a new application or update required development managers to submit a business case for the procurement of new hardware, wait weeks for delivery, then go through a painstaking process of provisioning and configuration. This has changed in recent years—first with the advent of server virtualization and more recently with PaaS providing complete cloud technology stacks for development and testing.

- *Containers*—A recent technology innovation, containers comprise an entire runtime environment, including the application, plus all its dependencies, libraries, and configuration files needed to run it—all bundled into one package. By containerizing the application platform and all dependencies, any differences in OS distributions and underlying infrastructure are abstracted away. Unlike virtual machines, containerized applications run a single operating system, and each container shares the operating system kernel with the other containers. This makes containers more lightweight and resource efficient than virtual machines. Containers can be run on public cloud services and are easily shared, which makes them particularly useful for development and testing teams.

- *Canary releases/dark launches*—This is a release method used to test the performance of software deployments and new functionality in a phased manner. Canary releases are an important aspect of agile development, since, by rolling out new features incrementally and testing them with real users, teams can gather feedback and implement improvements quickly.

- *Design for failure*—As businesses embrace public cloud services, teams are employing design methods to make applications completely

independent of the availability of underlying infrastructure. By building resilient systems in which inevitable failures have a minimum impact on service, design for failure allows teams to scale applications quickly while achieving higher levels of uptime, even during major cloud infrastructure outages.

Agile Empowerment Challenges

While many of the technologies and methods described above have empowered teams to deliver software faster, this doesn't guarantee success. While developers are tasked with producing great software code, they often neglect critical issues associated with application performance and supportability. In many cases the adoption of new technologies and practices exacerbates this problem due to some common failings:

Technology for Technologies' sake—It's understandable that developers want to use the latest tool, but this can lead to support problems. For example, opting for a NoSQL database because it helps avoid lengthy schema design issues may address short-term issues, but increases the support burden because IT operations and support have no experience with the technology. Operational cost structures can also be impacted, by way of increased training costs and hiring specialists.

> Tip
> Make new technology tool selection a collaborative exercise. Enterprise architects can coach teams on the importance of placing program or business level objectives above discrete tool requirement or personal preferences.

Misuse of New Technology—It's a fact of IT that new technologies often can and will be abused, unwittingly or intentionally. With modern technologies, lack of experience may entice people to use technologies in ways they were never designed. For example, blindly dictating that every monolithic and legacy application should be containerized whether they're appropriate for the technology or not.

> Tip
> Today's new technology is tomorrow's legacy. Always assess whether the use of technology is appropriate for the business and never underestimate architecture and design requirements.

Metric Misalignment—With developers using techniques like canary releases, dark launches, and split or A/B testing, businesses needs to know whether new features are successful and have led to more customer conversions and sales. Historically, however, IT operations have used technical diagnostics as a means to assess performance, which may be misaligned with the business.

> Tip
> Consider adopting monitoring methods that focus more on achieving desired business outcomes. In a mobile app context, this could include monitoring techniques to analyze app and functional usage by geographic area and user community. By managing to these outcomes, it becomes easier to assess the implications of any application performance issues and feed richer information and insights back to development.

Lost Opportunities—As new development platforms become increasingly abstracted from hardware infrastructure, many complex performance problems can surface. Because developers are shielded from the infrastructure, they can be slow to respond to any issues their code has introduced. This lack of insight also means they can fail to fully exploit the true potential of new cloud-based technologies.

> Tip
> In customer-centric computing, high-performance and low-latency are as important as functionality and design. Consider teaming junior and experienced developers with skilled IT operations and sysadmins to determine how modern hardware architectures can be better exploited to consistently deliver a high-quality customer experience.

Workarounds Due to Constraints—Even as teams look to adopt public cloud services and PaaS to accelerate development, they're still constrained by access to production systems for testing. This can involve delayed access to applications, infrastructure, and perhaps the most time-consuming and resource-intensive task in IT—creating, maintaining, and provisioning accurate, compliant and up-to-date production-like test data. With these constraints, teams may introduce sub-optimal testing practices that fail to detect software defects or compromise compliance with regulatory data protection requirements.

> Tip

> Consider technologies that can simulate unavailable systems across the development lifecycle. This allows developers, testers, and integration teams to work in parallel for faster delivery. Capabilities in test data management, including data subsetting and masking, together with synthetic on-demand test data generation, should also be assessed.

Modern Application Architectures

To support the businesses need to innovate, development teams must continuously deliver software services at an increased velocity. This has been recognized by web-native companies such as Netflix and Amazon, who've changed their software architectures to support the need for continuous innovation—essentially using them to redefine the markets within which they operate.

With agile methods, organizations can iterate quickly to support innovation, but teams are also changing application architectures to improve software flexibility and help accelerate deployment. For this reason, older style monolith designs are being supplemented with microservice designs (see Figure 2-3).
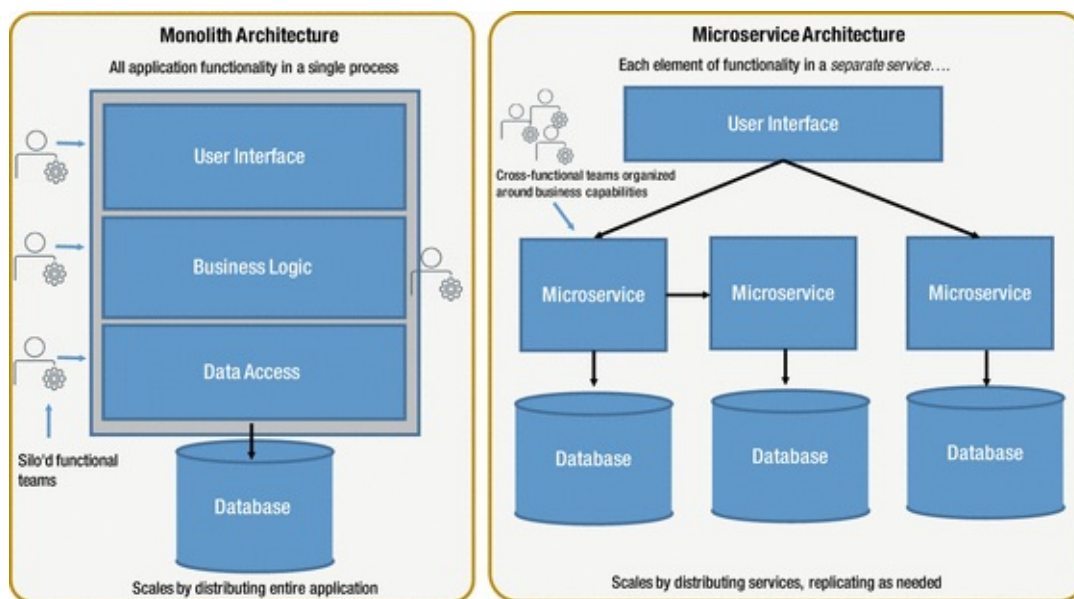

Figure 2-3. Monolthic and microservice application architectures

Until recently, application architectures were monolithic in design and operation. Although consisting of many services, monolithic applications are packaged and operate as a single unit. For IT teams now tasked with faster

delivery and deployment, the characteristics of monolithic design have presented a number of operational challenges:

- **Brittleness**—If any single application element of component fails, then the entire application fails. If a task such as payment processing consumes more CPU or memory, then the whole application can degrade.

- **Risk**—Because everything is packaged together (and fails together), teams may be hesitant to change supporting technology stacks and infrastructure. This can explain operational resistance to increased deployment rates, since even simple application updates to brittle monolithic applications could cause system-level outages.

- **Tightly coupled**—Applications can only be scaled by deploying the entire application on more servers. This can be highly problematic in new mobile application development scenarios when demand is difficult to predict.

- **Dependencies**—Since applications are tied together, developers may have difficulty working independently to develop, test, and deploy their own software components. Development time increases and productivity suffers because they're often dependent on other teams finishing work before they can start.

Microservice designs differ from monoliths in that they involve building applications as a set of small, independent services. In essence, each microservice focuses on a specific element of business functionality. For example, in a web application there could be many microservices supporting everything from login processing and shopping carts, recommendation services and payment processing. Loosely couple in nature, microservices communicate with other services via Application Programming Interfaces (APIs).

Microservices can provide business a number of significant benefits:

- **Independent deployment**—Scaling becomes less problematic. For example, in a web-based shopping application a team can quickly deploy the instances of service it needs to meet demand spikes, while scaling back others.

- **Independent coding**—Teams have more freedom to develop in different programming languages, each optimized for different processing tasks. Microservices can free organizations from being locked into a single technology stack.

- **Fault tolerance**—When a microservice fails, it's unlikely that the entire

system will fail. If the recommendation service fails in a web application, shopping cart and payment processing services can continue to function.

- *Increased agility*—Microservices designs can better support continuous delivery. Since systems are built and deployed independently, they can potentially be tested and delivered faster.

Microservices: Small Isn't Always Beautiful

While the simple and elegant nature of microservice style design delivers many benefits, substantial complexity exists in terms of management and operations. If ignored, these issues could increase friction between development and operations teams. Important considerations include:

- *Supportability*—The support burden can increase substantially—especially when IT operations are suddenly faced with managing hundreds of microservices developed in different languages, accessing new data stores, and running on cloud platforms and infrastructure.

- *Monitoring*—Managing a single monolithic application is demanding, but now IT operations has to ensure many more processes remain performant. With highly distributed microservices systems, a whole new set of management considerations surface. These include network latency, fault tolerance, asynchronous messaging issues, and network reliability.

- *Coordination*—Deploying hundreds of microservices demands rigorous deployment processes that can be beyond the capabilities of teams using manual processes and scripting.

> Note
> Modern management approaches to address microservice deployment and monitoring challenges are discussed further in Chapters 6 and 7.

Ending the Technical Impasse

Even with advances in tooling, development methods and software designs, the friction created by teams operating in discrete functional silos can negate all potential benefits. Fractured processes across the application software lifecycle inevitably result in slow delivery, low productivity, and defect-ridden software systems. This has to change.

IT's value proposition isn't just to keep the technology lights on and periodically deliver improvements over long cycles; it's to continuously manufacture business value from a modern high-performance software factory.

DevOps, with its focus on close collaboration between development and other IT groups, while automating essential application delivery processes, is now a critical business imperative.

Summary

The following chapters outline key strategies that can help IT and digital leaders accelerate a successful and business-aligned DevOps initiative.

Starting with Chapter 3, we'll describe how to build a winning DevOps culture and re-energize the IT organization. Here, we'll examine easy and cost-effective ways to increase collaboration, the application of Lean thinking to reduce IT waste, and what constitutes a comprehensive DevOps metrics program.

3. DevOps Foundations

Culture, Lean Thinking, Metrics

Aruna Ravichandran[1]✉, Kieran Taylor[2] and
Peter Waterhouse[3]

(1) CA Technologies, Cupertino, California, USA
(2) Winchester, Massachusetts, USA
(3) Blackburn, Victoria, Australia

Blink during a Formula 1 pit-stop and you'll probably miss it. But this wasn't always the case. Fifty years ago, a pit-crew would take over a minute to change the wheels and refuel. Today, anything more than three seconds is considered a fail.

It's the same in software development, where teams once tasked with updating enterprise applications at a sedate pace must now deliver new software services as a continuous flow of value to customers.

The problem for today's enterprise, however, is that software teams don't work like Formula 1 pit-crews. Rather than working in tandem, IT teams often work serially—development codes, then QA tests, and finally IT operations monitors. However, with application software released, enhanced, and retired over more compressed timeframes (months and even days), this stop-start method of development falls short. It's as ineffective as each member of a Formula 1 pit-crew replacing a tire and checking wheel nut tension before the next one could start—the race would be over before the car left the pits.

While we can celebrate the heroics and skill of great racing car drivers, what sets successful constructors apart is their ability to build a winning culture irrespective of role and responsibility, be that driver, team manager, telemetry engineer, or aerodynamics chief, everyone is focused on a singular goal—

winning races. It's why drivers thank the teams before they spray champagne on the podium.

Like Formula 1 drivers, technological advancements have improved the efficiency and effectiveness of IT professionals. However, in organizations that traditionally measure and incentivize based on technical specialization within functional areas, relying on tools alone will never build the collaborative culture needed for business growth and profitability.

What Characterizes DevOps Culture?

DevOps is very different from traditional thinking because it places great emphasis on culture. It instills a shared sense of vision across multiple teams, directly aligned to the business and its customers. To this end, maverick behavior, such a cutting corners and allowing defect ridden code to go into production, or blaming operations when a software release fails, is counter to a DevOps thinking. With DevOps unified IT is the hero and no one is singularly to blame for problems.

But this is challenging in IT because of the friction existing between development and other IT teams—especially IT operations. On the one hand, developers are focused on accelerating change by faster delivery of applications, while the operational mantra has been resilience and stability at all costs, even if that means holding back change.

Evidence suggests, however, that while both these goals are equally important, they are not mutually exclusive. For example, the 2016 Puppet Labs "State of DevOps" report illustrated that high-performing IT organizations are well able to achieve faster software delivery along with increased resilience and stability.[1] Clearly, DevOps high performers have ended the divisional "turf wars" by enacting strategies to re-shape entrenched silo thinking and behaviors into a more powerful collective force.

Since DevOps culture involves creating new shared values and behaviors across IT teams, leadership must play an active role in driving these characteristics across the entire organization.

Focusing on Products over Politics

Traditionally, IT teams have been organized in technical silos. Interaction and communication has been conducted through overly engineered and rigid processes. Software changes run the gauntlet of lengthy change-management processes, human intervention, and change review boards. Though not wrong

per se, these elements were designed to cater to situations where change was less frequent but occurred in greater volumes, requiring more rigor to ensure operational stability and compliance.

A strong DevOps culture, however, is characterized by systems thinking. That is, a collective emphasis on service as a whole, not on discreet functional elements or processes. Rather than persist with technical fiefdoms, DevOps aims to break down barriers—organizing by product over structure and continuously driving improvements in context of a products lifecycle, from the inception of an idea to full production status. Strong leaders recognize this by promoting open communication, using shared metrics, and establishing (even automating) feedback mechanisms within and across teams.

Building Trust and Respect

Over many years, respect has been garnered by individual contributors. Be that superhuman developers who crank out code, or on-call operations staff who fix problems at 4:00am. In a thriving DevOps culture, hero worshipping takes a back seat to collective respect. With DevOps, everyone should respect the contributions of others and no one should be afraid of speaking up for fear of abuse and vilification.

This is critical, because from healthcare to aerospace, studies have shown that bad practices and behaviors can over time become accepted as normal practices—often with disastrous consequences (see Chapter 8 for further discussion on strategies to combat normalized bad practices). In IT this happens all the time due to power games and lack of respect. Even if new staff witness blatantly suboptimal practices, they'll be loath to report it for fear of rebuke and retribution by managers, eventually accepting the situation and practicing it themselves. DevOps leaders should be mindful of this and work across the product lifecycle to identify situations where violations are tolerated because people are afraid to speak up or look mean.

Trust also plays an important role in DevOps culture. Just as the Formula 1 driver trusts his pit-crew to fit four wheels securely, so must cross-functional trust be established across IT. For development, this means trusting that the production performance information from operations can actually help in software refactoring and reducing technical debt. For operations, it means trusting new application design patterns will help the business scale. Everyone from security to enterprise architecture is part of the trust equation, and as the speed of software delivery accelerates, no DevOps program will be successful without it.

Increase Empathy Everywhere

It's well understood how important the role of empathy plays in today's app-centric software design. Without understanding the emotional and physical needs of customers, together with their behavioral patterns, businesses risk substantial losses from their software investments. This explains why many organizations conduct rigorous design experiments before any full software release. This is illustrated in the extreme by Google's "50 Shades of Blue" user interface testing exercise.[2]

Yet despite this, empathy is lacking within many enterprise IT departments. Teams usually operate in separate locations, so development and operations teams have few face-to-face opportunities necessary to share each other's pains, surface concerns, or raise issues.

There are many simple but effective strategies leaders can use to build empathy. Not the least this should include building closer ties between development and support. Even with the greatest software and delivery processes it's important to understand their perspective and what they experience when dealing with customers.

> **Tip**
> When developing products or new features, put yourself in the position of the customer and support staff by examining all the situations where they might need help.

Staff (including developers) should understand the importance of enabling a great customer experience. To that end, consider working directly with customers directly in the field in a variety of adverse situations. What happens when as a customer you're trying to board an airplane and the scanner breaks? Or what's the impact when a mobile app crashes or bad network coverage means you can't call road side assistance?

Obviously it's not always practical or feasible for developers to work this closely with clients; however, with analytics tools, staff can put themselves in the "shoes" of the customers and gain realistic insights into the customer experience.

Open Communication Channels

In 1968 a computer programmer, Melvin Conway, postulated that organizations that design systems are constrained to produce designs which are copies of the

communication structures of these organizations.[3] In a DevOps context, what's now dubbed Conway's Law has great relevance, especially in situations where organizational structures and closed communication channels prevent developers and operations from agreeing on IT performance objectives (e.g., increased change frequency and improved reliability). In such cases, it's possible for team-based activities to be prioritized over more important cross-functional improvement strategies.

There are many possible solutions to this problem. The technology teams at Netflix and Amazon structure themselves around small teams, with each one responsible for a small part of an overall system. Spotify promotes collaboration across team boundaries via agile development squads, chapters, and guilds, with a separate IT operations team providing all teams the support needed to release software themselves.

Looking beyond technology-centric companies, there are other examples of businesses thriving because they've modified communication structures. Take Zara for example.

Year after year in the fickle world of retail fashion and apparel, Zara continues to increase revenue and profit. For Zara, flexible responsiveness to customer demand, backed by a tightly integrated supply chain, is fueled by teamwork and collaboration. In retail stores, managers use real-time intelligence to place orders and feedback information directly to the point of manufacture. Different teams (including design, product management, and merchandising) use shared spaces and work closely together. With increased emphasis on communication from initial garment design to distribution to the shop floor, Zara has shortened product lead times and doesn't unnecessarily commit large volumes of product in advance of a fashion season.

Considering approaches like these, cross-functional IT collaboration could be improved when:

- Leaders apportion budget to practical co-location strategies. This should be more than simple staff re-housing and include shared work spaces and lounges, together with team huddle areas and common wall whiteboards.

- IT operations team members regularly participate in agile standup meetings in order to appreciate the value of deploying code quickly and how their current activities help or hinder release processes.

- Development teams attend operational post-mortems or workshops to gain better insights into the problems caused by poorly performing or insecure software.

- IT operations works with developers to establish performance monitoring in pre-production so as to detect problems earlier where they are easier and less costly to fix.

- Developers are placed on the after-hours support roster to better appreciate the impact of problematic software code on users and customers.

- Support specialists share critical application experience analytics obtained during mobile app engagements with customers.

Additional Factors

Changing IT culture isn't easy. Right or wrong, people have pre-conceived notions and firmly entrenched ideas. Any sudden shift in workplace practices and it's only natural that people will feel threatened and push back. Addressing this means patiently working with people to enact the necessary behavioral change or move people to different jobs.

With DevOps and cultural change, it's critical to start with a clean state. Rather than dive head first into massive IT workforce transformation programs, leaders should first assess the cultural landscape from a business perspective. This involves understanding the primary goal of the business and then analyzing whether prevailing behaviors support it. Although seemingly obvious, many organizations miss or neglect this critical step. If, for example, a company defines its goals too broadly, people working in different IT teams will interpret them in different ways and shape activities accordingly, often to the detriment of each other and the business.

In a broader sense, culture will also be influenced by an organizations' business model and operational perspective. There are three classes to consider:

Run the business—In this model the overarching strategy of the business is on continuing operations in much the same way—only better, faster, and cheaper. Here the focus for IT is operational excellence—adopting new technologies, yes, but using them to pound out efficiencies across processes like warehousing and logistics.

For these organizations, IT culture is characterized by discipline and rigidity—all fine to support efficiency improvements, but inadequate in a world where old business rules are constantly being challenged by disruptive technology.

Grow the business—The business strategy shifts from doing more of the same to conducting the same business in radically different ways. Netflix

presents a good example of this model. Five years ago, Netflix delivered DVDs through the mail, now they stream entertainment over the web— even creating their own content. Customers still turn to Netflix for entertainment, but the way in which Netflix is addressing that need has fundamentally changed.

For organizations in this category, the culture also needs to change. For Netflix, their DVD-delivery model required strong operational oversight over processes like inventory management and distribution to drive efficiencies and increase customer satisfaction. Now however, their streaming model and content generation programs requires teams rapidly delivering new services based on quickly analyzing customer preferences and optimizing web performance and throughput. Obviously if a company's cultural behaviors and values are still skewed toward driving efficiency in one operational area, pivoting to the new model will be difficult.

Transform the business—This model carries the most promise and risk because it involves changing the very fabric of a company. Businesses don't just conduct the same types of business in new ways, they reinvent themselves completely. For Amazon that's meant moving to being a mega cloud computing provider from selling books. For Walgreens, it's meant going from selling medicine over the counter to treating illnesses in stores.

With strategic transformation examples like these, the business is introduced to new dynamics and competitors. Now, IT performance will not only be judged on growing the business of today, but also on creating the core business in the future. To this end, a DevOps culture built on open communication and collaboration, trust, respect, and empathy isn't just important for short-term growth, it's essential for long-term business sustainability.

   Once business models and goals are clearly understood and communicated, any required IT behaviors and values needed to support them can be influenced through the development of fully aligned IT goals and metrics. These could include shortening lead times for new products to support faster time-to-market, increasing mobile app customer conversions to support increased revenue objectives, or helping the current business scale by making better use of cloud infrastructure.

Lean Thinking to Reduce Waste

Fuel strategies play a significant role in Formula 1 racing. A car with a half-full tank can be as much as three seconds faster than a rival vehicle with a fully loaded fuel cell. Extra fuel equals extra weight, so teams go to great lengths to calculate the exact amount of fuel needed under full race conditions.

Though beautifully engineered and continuously refined, racing car engines are still flagrantly wasteful. Like your car at home, the dynamics of internal combustion still mean that only a certain percentage of fuel stored in the tank is converted into useful energy. The rest is lost as heat and friction and explains why teams constantly refine chassis designs to reduce aerodynamic drag.

Beyond the frenzied world of Formula 1, many elements contribute toward engine waste in the cars we lesser mortals drive. Running an air conditioner consumes fuel without contributing to motion. Friction in engine pistons wastes fuel, as does tire pressure and extra luggage. All told, as little as 14 percent of passenger car fuel is converted into useful energy. Clearly, this gas guzzling engineering throwback is rife for disruption from electric cars and advances in battery technology—time will tell.

In many ways software development is as wasteful as the internal combustion engine. Friction between development and operation causes delays. Manually assembling multiple components and configurations within our own software factories leads to lost time. Carrying excess inventory in the form of unneeded infrastructure capacity adds extra cost. Add to this constraints preventing access to critical systems and data during testing, and defects can accumulate across the software lifecycle.

Lean and Value Creation

The traditional view of IT value has been internally-shaped. For decades, systems and applications have been designed, built, tested, and released to customers, citizens, and end users where they hopefully influenced behaviors. All this has changed. With the advent of cloud, mobility, and social computing, consumers rather than producers call the shots. This means businesses find themselves in the position of having to respond to the behaviors and desires of their customers.

For IT, this redefinition of business value means teams must focus on two essential strategies. First, they must continuously reexamine the software services they deliver from the perspective of the customer, and second, they must constantly strive to minimize any interference or waste across the entire software factory. This includes everything that impedes the flow of value to customers and incurs more cost.

This notion of value being "pulled" by customers and waste elimination is not new. Lean pioneers and practitioners such as Toyota, Motorola, and Xerox redefined manufacturing by applying these principles; understanding that many forms of waste exist across production processes. And, because they add no value to customers, must be clinically removed.

But can Lean principles be applied in IT, where, unlike traditional manufacturing, waste isn't visible across a factory floor through telltale signs like excess physical inventory or idle machinery? Often due to its intangible nature, waste in IT can be hard to identify yet alone eliminate.

Interestingly, the delivery of software bares many similarities to a manufacturing process. In IT, we have the means to respond to value triggers from our customers by quickly designing, developing, and releasing software services. And, since the software delivery lifecycle represents a manufacturing production line within a software factory, we have the guiding context upon which identify all elements of waste that add no value to the business and its customers.

Eight Elements of Waste

As illustrated in Table 3-1, there are eight elements of waste or "Muda" (using Lean terminology) that severely impact the IT group's ability to increase the value of software services.

Table 3-1. Eight Elements of Waste (D.O.W.N.T.I.M.E)

Type of Waste	Examples	Business Outcomes
*D*efects	Badly designed and poor quality code Non-functional performance issues	Lost customers and revenue; negative brand impact
*O*verproduction	Delivering features customers don't need or want Procuring extra capacity due to unanticipated performance requirements	Delays, cost over-runs, and budget problems
*W*aiting	Excessive release backlogs and bottlenecks Infrastructure and data not available for testing Change reviews; security and compliance audits	Slow time-to-market and value; lost opportunities
*N*on-value added processing	Lengthy problem resolution and fire-fighting Team-based activities prioritized over program-level objectives	Morale issues; high-staff turnover
*T*ransportation	Frequent release rollbacks Development/QA handoffs	Application launch delays; increased cycle times
*I*nventory (excess)	Underutilized resources Partially completed work and excessive work in progress	Increased capital and operational costs

*M*otion	Developers constantly switching Relearning and rework	Lost productivity; talent erosion
*E*mployee knowledge (unused)	Closed retrospectives and stand-up meetings No feedback established from service management (e.g., call center/service desk)	Missed opportunities to drive improvements

Examining this table it should be noted that there are close relationships and linkages between elements. For example, undetected code **defects** resulting in performance problems may result in an organization purchasing additional hardware capacity, which leads to excess **inventory**, which increases the support burden. In situations like this, waste begets waste and technical debt accumulates to such an extent it becomes difficult to pay off. The result is that essential development is tied up on maintenance and support activities.

Originally coined by Ward Cunningham in the Agile Manifesto, technical debt has tended to be reviewed from a development perspective[4]. After all, if software defects can be identified and eradicated during early stages of development, then production related problems (which could be significantly costlier to fix) can be avoided.

But technical debt can also be created in IT operations. For example, failing to document or visualize business services (and supporting applications an infrastructure) means teams could take longer triaging problems. Here the waste is **non-value added processing**, which again (because of linkages) results in more waste. In this case, increased **transportation** because a release has to be rolled back.

Using the eight elements of waste list, DevOps practitioners can begin a process of identifying waste elements across the software lifecycle. It's important to understand that "toxicity" levels will vary, so mechanisms must be developed to continuously reveal new situations and conditions that can potentially introduce more waste.

It's also critical not to restrict the exercise to new development. These may become the debt burden of the future, but could only represent a small part of the portfolio. Legacy infrastructure and production applications should also be included because, even though they change less frequency, they often incur significant management costs and overheads.

Finally, debt and the associated waste should be reviewed as a continuum, with special attention paid to integrations between new customer facing apps and essential back-end business processes. For most enterprises, multi-channel engagement creates tremendous opportunity for value creation, but will introduce more waste if they're not integrated and coordinated with existing

back-end systems, applications, and call-center services.

Waste Removal Strategies

Today's mobile and API-centric forms of service delivery mean that customers assess value based on extremely high levels of functional and operational quality. They also expect businesses to deliver additional value in the form of continuous change. With customer experience so important, it's critical to begin waste identification from the perspective of the customer; monitoring and analyzing the usage and behaviors of applications and determining what elements impact the total experience. This is especially important for mobile applications, since factors beyond the control of IT departments (e.g., carrier network latency and cloud service performance) can quickly erode value, however good the functional quality.

Some immediate practices cross-functional teams can apply to help identify and eliminate waste, include the following.

Prevent Defects by Removing Constraints

When development and testing is constrained due to lack of access to dependencies (e.g., middleware, web services, and test data), defects can quickly work their way into the code base. This is illustrated in a Service Virtualization survey, which identified that on average, participants require access to 52 dependent elements for development or testing, yet have unrestricted access to only 23 of these.[5]

To circumvent these issues, many development teams often attempt workarounds by hand coding (mocks and stubs), but this doesn't provide for realistic application behavior, causing test validation errors and the late discovery of defects. Discussed further in Chapter 5, a more scalable approach is to incorporate Service Virtualization into parallel development and test activities.

Focus on Value to Prevent Overproduction

New application features don't necessarily mean more customer conversations and increased revenue. Unnecessary features can result in additional maintenance overheads and cost. There are many methods DevOps practitioners can use to reduce this form of waste, including:

- Incorporating application experience analytics into monitoring strategies to

identify mobile app functions and features that are not used

- Split or A/B testing and funnel or cohort analysis
- Refactoring code elements to reduce complexity, remembering that the cheapest and most reliable components are those that don't exist!

Smoothing Flow to Reduce Wait Times

Like waste element #1, this waste can eventuate due to delays waiting on dependencies during development and testing. In the VOKE report mentioned above, 81 percent of participants identified development delays of waiting for a dependency in order to develop software, reproduce, or fix a defect. Additionally, 84 percent of participants identified QA delays of waiting for a dependency in order to begin testing, start a new test cycle, test a required platform, or to verify a defect.

> Caution
> Never underestimate the wait times associated with accessing test data, since a massive 20 percent of the average software delivery lifecycle is wasted waiting for data, locating it, or creating it manually when none exists.

Excessive wait times may also be due to problems managing highly complex release and deployment processes. However good the code, its ultimate value will be determined by how quickly it can be deployed into production. Manual processes and fragile scripting not only compromise these goals, but also increase the potential for defect code being released. These issues can be addressed by:

- Ensuring all key stakeholders possessing the knowledge to move a service swiftly across lifecycle are involved early and often
- Using smaller batch sizes so that value is delivered to customers at regular intervals
- Developing and automating reusable and repeatable processes to simplify and streamline application releases

Limit Non-Value Added Processing Through Data-Driven Insights

Fixing application problems provides limited value to customers. Rather than

wait for problems to occur in a production, IT operations should be involved much earlier in the development lifecycle.

Using tools to share information is especially valuable. By leveraging application performance change impact analysis during a build process, for example, developers can quickly determine any adverse performance conditions their code is introducing.

Reduce Transportation Cost by Automating Deployments

When work is manually handed off from one team to another (e.g., developer to test/QA, QA to operations), critical knowledge can be lost. This could lead to additional delays or the highest transportation cost of all—release rollbacks.

There are many strategies to address these issues, including:

- Reducing the number of handoffs by automating standard tasks and activities

- Ensure release automation tools provide an extensive set of action packs and plug-ins so as to fully deploy at an application level, while also integrating key supporting processes (e.g., configuration management)

- Build more knowledge as releases progress (e.g., establishing application performance management in pre-production)

Eliminate Excess Inventory Across the Software Factory

Minimizing inventory is the hallmark of Lean thinking. As in traditional manufacturing, there are many waste indicators in IT's own software factory. In development, partially completed work can become obsolete before it finds its way into production and should be exposed to ensure it doesn't degrade or corrupt the code base. In operations, excess on-premise server infrastructure acquired as a fail-safe to address unanticipated performance problems could be avoided by establishing monitoring in pre-production.

> Note
> Costs can accumulate substantially when agile teams acquire specialist tools. Work collaboratively to assess whether the additional cost (training or support) offsets the value delivered to one team.

Prevent Unnecessary Motion with Parallel Development

While transportation waste is associated with the unnecessary movement of software, motion waste involves the unnecessary movement of people. A good example is task switching, where an API developer might shift focus to a new project rather than wait for testing dependencies to become available.

Apart from adding more waste (e.g., delays), task switching can introduce many more problems, especially related to the productivity of developers due to constant interruptions.

Some simple strategies to reduce this waste, especially task switching, include:

- Try to ensure teams have all of the knowledge, tools, and data needed to complete their assigned work

- Simulate and virtualize all dependencies so that development teams can code and test in parallel

- Since as much as 50 percent of testing is wasted by teams trying to locate test data or create it manually, consider supplementing constraint-removal strategies with test data management (see Chapter 5)

- Aim to eliminate unimportant work, meetings, and interruptions. If it isn't delivering value, ask why your team is doing it!

Incorporate Employee Knowledge Using Feedback Loops

While the feedback of production information is important to drive software improvements and improve supportability, it isn't the only place where knowledge can be transferred.

Service desks and call-center processes should also include mechanisms to deliver (to development) important information gained from customers on their usage and response to new application features and functions. Knowledge transfer should also be bi-directional. For example, application experience analytics could (when integrated with incident management processes or even social media) become an early warning mechanism to trigger coordinated responses in the event of mobile app usage problems.

DevOps Metrics

With any IT-driven methodology or program, measuring the effectiveness in a business context is critical. But since DevOps isn't a formal framework, organizations have little guidance in determining what metrics should be used.

This can be problematic and lead to a number of suboptimal practices:

- ***Efficiency status-quo***—The IT team falls back to metrics traditionally used to demonstrate technical proficiency in meeting stability and resilience goals. Although these are not necessarily wrong, DevOps metrics should also demonstrate how new processes and automated technologies are impacting the business—for example by speeding time-to-market and reducing lead times.

- ***Outputs over outcomes***—Organizations gravitate to metrics that are commonly used in assessing team-level productivity. These can include output-based metrics like number of features delivered or servers provisioned. Metrics in this class can be counterproductive unless balanced with outcome-centric indicators that show results achieved against desired quality levels.

- ***Low-hanging fruit***—Organizations select metrics that are easily obtained but not necessarily useful. Since DevOps success is predicated on cultural change, businesses must also measure what's harder to determine but potentially more valuable—namely, how the adoption of DevOps behaviors and values at an organizational level is impacting the business.

Anti-Pattern Metrics

Before embarking on a metrics refresh, organizations should consider all existing measures and their applicability in a DevOps context. Particular attention should be given to carefully review those metrics and incentives that are counter to DevOps principles, as illustrated in Table 3-2.

Table 3-2. Problematic Metric Classes

Metric Class	Examples	Adverse Effects
Vanity Metrics	Lines of code produced Function points created	May be counterproductive since they reward the wrong types of behavior—especially if incentives are linked to the metric. Producing more code and features without validation can inhibit other valuable activities such as refactoring and design simplification.
Intra-Team Metrics	Agile team leaderboards Deployments/changes prevented	Beware of metrics that pit-teams against each other and use vanity metrics as scoring mechanisms. Strike a balance with metrics and rewards that influence positive inter-team behaviors—such as code sharing, peer reviews, and mentoring. Pay particular attention to metrics that promote an anti-DevOps culture, such as rating operational effectiveness on the ability to prevent releases and deployments.

Traditional Metrics	Mean-time-between-failure (MTBF) FTEs: Servers	With faster delivery of services, some failure is to be expected. Always consider that improving responsiveness can be more important (and less costly) than trying to prevent failures.

Suitability Checklist

When reviewing and developing DevOps metrics, it's also important to consider each against a general suitability checklist:

- *Obtainable*—Culture and behavioral improvements are important to measure, but metrics may be difficult to obtain or quantify. Seek out other related data points to help expose —e.g., staff retention rates/transfers as an indicator of employee morale.

- *Reviewable*—Every metric must stand up to rigorous scrutiny in a business context. Carefully review metrics that can be easily collected, but add no tangible value—e.g., lines of code produced per developer.

- *Incorruptible*—Determine whether each metric can be influenced by team and employee bias. Seek out any associated incentives that can work against a collaborative DevOps culture—e.g., existing SLA bonuses inhibiting change.

- *Actionable*—Any metric must support improved decision making. Exposing A/B testing results can for example be a valuable way to quickly determine the effectiveness of new functionality.

Wherever possible, metrics should also be shareable and have relevance across the software lifecycle to both development and operations. For example, generating security scores at a cross-functional team and divisional level can be used to inform teams about the risks of their actions.

Metrics that Matter

Having determined what not to measure, the next stage is to develop a candidate list of metrics supporting the DevOps program. One common mistake is to measure too many elements, falling back to what's easily collectable. Additionally, metrics applicable to DevOps may be new to organizations (e.g., the speed of deployment, rate of change, and customer responsiveness), so it's important to think broadly how changes to work practices, process and technology can support these goals.

- *People*—Staff related metrics can be the most difficult to collect but are

still powerful change indicators. Strong consideration should be given to internal metrics like staff retention rates and training, together with mentoring and knowledge building (e.g., open source contributions and wiki development).

- *Process*—It's important to consider how existing practices will help or hinder new targets being achieved, paying special attention to existing bottlenecks (e.g., security audits only conducted after testing will impact deployment rates).

- *Technology*—Good metrics are those that help teams drive improvements, even after failures (e.g., what is the percentage of failed releases and what percentage of these were due to code defects, manual processing, configuration errors, etc.).

When developing metrics, it's important to maintain balance. Defaulting to metrics skewed toward one particular area (e.g., operational or development efficiencies) can have a negative effect in terms of behavioral improvement

Figure 3-1 illustrates four dimensions and sample metrics that can be used to measure the effectiveness of a DevOps initiative.
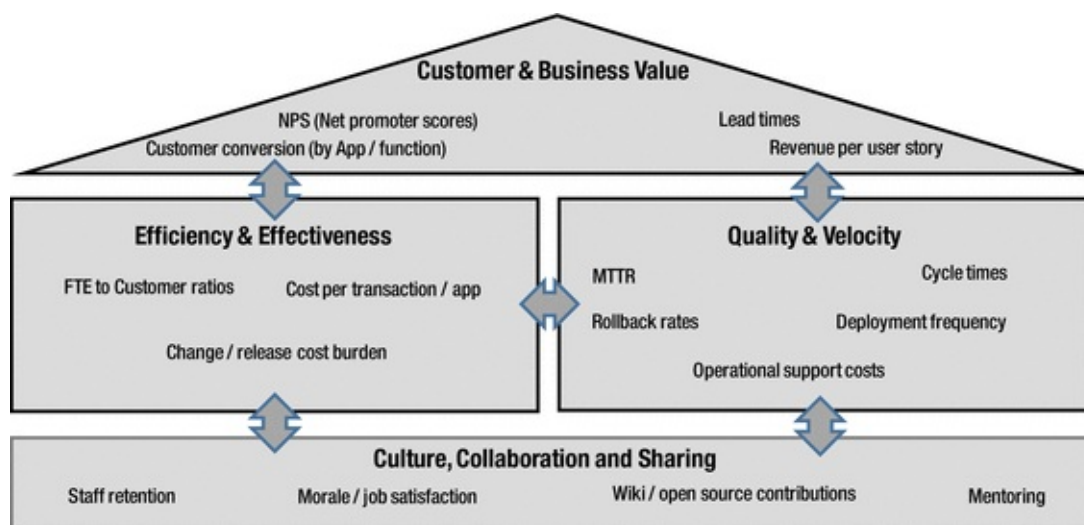

Figure 3-1. DevOps metrics dimensions

Culture, Collaboration, and Sharing

Metrics in this category are especially valuable because they provide an ongoing indicator of acceptance/resistance to DevOps. Some metrics in this dimension will be easier to collect (e.g., staff retention rates/turnover) than others (e.g.,

employee morale). It's important therefore to look at measures across other dimensions to understand how they impact this area. For example, are mean-time-to-recover (MTTR) improvements positively impacting staff morale, absenteeism rates, and responsiveness to change? Consideration may also be given to automated surveys and employee feedback, as long as these are fully transparent and actionable.

Efficiency and Effectiveness

Metrics here normally focus on elements of development capacity and operational capabilities. While traditional metrics such as server to sysadmin ratios have been used, many organizations are now adopting more customer-centric ratios like full-time-equivalent (FTE) to customers.

Examining full costs on a transactional or application basis is another good candidate metric, as it's focused on improving data center efficiencies (e.g., energy and cooling). Other metrics such as cost of release are also good since these can expose inefficiencies associated with acquiring, preparing, and maintaining physical infrastructure for development, testing, and production.

Quality and Velocity

This dimension looks to measure data points with respect to service delivery. For organizations starting on a DevOps initiative, many indicators (e.g., percentage of deployments rolled-back due to code defects/outages/negative user reactions) could initially be high. This may be a result of the extra time needed to adopt new processes, combined with remediating existing technical debt and waste elements these metrics expose. However, with DevOps' focus on establishing quality right from the start of development, this should reduce over time.

When paired, these metrics also provide additional insights. For example, if the **rate of rollbacks** still increases during periods of low **change volume** it could be indicative of serious problems, e.g., errors due to manual/scripted release processes, task switching, and excessive handoffs.

Other useful metrics in this dimension include:

- **Cycle time**—Measures the length of time it takes to complete a stage or series of stages in a release operation. This can be extremely valuable in exposing any bottlenecks.

- **MTTR**—This can be broken down into detection, diagnosis, and recover phases. MTTR is a great indicator of how effective teams are in handling changes. For complex deployments, there will be spikes, but this metric

should be trending down as DevOps becomes established.

Customer and Business Value

This category of metrics are externally focused and help measure how DevOps supports business goals—like increased customer loyalty and faster time-to-market. The manufacturing concept of lead time provides DevOps practitioners with an analogous metric (time taken from when code starts development to successful production deployment) and determines how well DevOps is at meeting the need for rapid delivery of high-quality software services. This metric is especially important to scrutinize because long lead times could be indicative of code defects or testing constraints.

Another interesting candidate is Net Promoter Score (NPS), which is a simple management method to measure customer loyalty. While this metric has traditionally been used in other areas of the business (e.g., marketing), its inclusion is valid since the loyalty of customers is increasingly determined by how quickly high-quality software services and updates can be delivered to via web sites and/or mobile apps.

Additional Methods and Techniques

With metrics developed across each of the four dimensions discussed previously, teams can begin a process of determining the relationships between them. This is important so teams gain insight into what processes enhancements and tools are needed to meet targets or address capability gaps.

One simple and effective approach, as illustrated in Figure 3-2, is business impact mapping. This involves determining which DevOps processes will be needed to support a business or customer experience goal, together with the underpinning metrics, targets, and initiatives/tools across multiple dimensions that support this outcome.
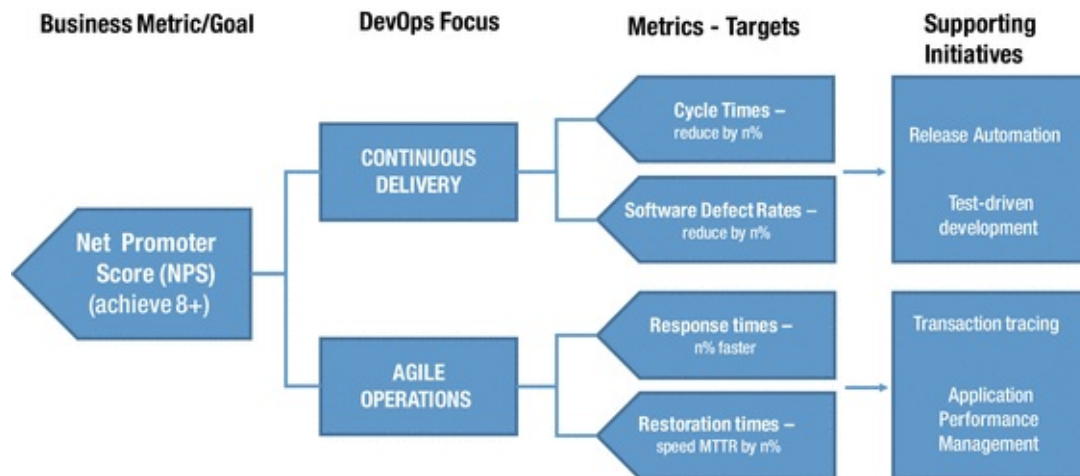
Figure 3-2. Metrics, targets, and initiatives linked to business outcomes

Figure 3-2 illustrates that an organization is seeking to achieve a Net Promoter Score of 8+. To support this goal, IT needs to deliver software releases and new functionality faster, together with ensure a high-quality customer experience. Metrics and targets have therefore been set within the quality and velocity dimension, together with targets and supporting process/tool initiatives.

As DevOps metrics programs develop, practitioners should also:

- Have regular and ongoing target reviews to ensure that goals are not completely unrealistic or that existing processes and tools are not delivering improvements.

- Consider removing persistent "green light" metrics when targets have been consistently achieved.

- Avoid having every metric focused on velocity without paying attention to customer satisfaction and loyalty.

- Strive to prevent vanity metrics and operational or team-centric bias creeping back into the program and distorting the true performance picture.

- Beware of ranking teams based on targets—the best way to compare teams is to measure things like customer loyalty (as described) and how successful teams are in meeting their commitments.

- Give strong consideration to metrics, targets, and initiatives that foster peer review and openness.

- Carefully build incentives and reward programs that reinforce the value of a strong collaborative culture.

- Involve business counterparts right from the start to ensure customer and business data is held to the same standard as operational/efficiency metrics.

- Match tools to the DevOps program, especially those that can monitor and respond to real-time conditions (such as transaction times, response times, and mobile app crashes), but can also proactively detect and prevent adverse conditions (such as code defects and release bottlenecks) that impact performance.

Summary

At its heart, DevOps is about building a generative organizational culture where business improvement is placed above everything else. But as this chapter has illustrated that won't always be straightforward, especially in organizations beset by divisional friction and lack of direction. By leveraging this chapter's guidance, especially with regard to building high-trust teams, an outcome-based metrics program and Lean thinking, organizations have a solid foundation upon which to guide their DevOps programs.

In Chapters 4-7, we'll look closely at the automated tooling needed to support this goal and how businesses can refit and re-engineer their own software factories to manufacture high-quality software innovations, at speed. In these chapters, we'll examine critical tooling strategies across the software lifecycle continuum—Build, Test, Deploy, and Manage.

Footnotes

1 https://puppet.com/blog/2016-state-of-devops-survey-here

2 https://www.theguardian.com/technology/2014/feb/05/why-google-engineers-designers

3 Melvin E. Conway, "How do Committees Invent?," *Datamation* 14 (5) (April 1968): 28–31.

4 The Agile Manifesto: http://www.agilemanifesto.org/

5 VOKE Market Snapshot™ Report: Service Virtualization;
https://www.ca.com/au/register/forms/collateral/voke-market-snapshot-report-service-virtualization.aspx