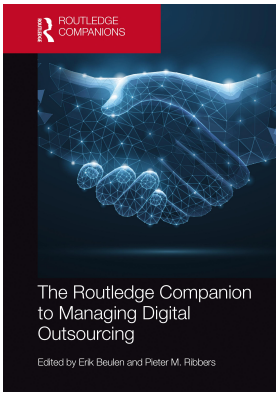

The Routledge Companion to Managing Digital Outsourcing

Erik Beulen, Pieter M. Ribbers

Agile and DevOps in the context of enterprise architecture and IT architecture

Publication details
https://test.routledgehandbooks.com/doi/10.4324/9781351037785-5
Bill Schiano
Published online on: 28 Jul 2020

4

AGILE AND DEVOPS IN THE CONTEXT OF ENTERPRISE ARCHITECTURE AND IT ARCHITECTURE

Bill Schiano

4.1 Introduction

Bharadwaj [1] describes the role of digital business strategy as a fusion of formerly functional-level IT strategy and business strategy. Gartner [2] disputes this assumed elevation of IT, arguing that for over 30 years similar elevations had been expected following innovations in technologies with business potential, but did not materialize in most organizations. Whether or not IT is elevated, it must still continue to drive and support the new strategy along the four dimensions Bharadwaj outlines: scope, scale, speed, and sources of value creation and capture.

As Bharadwaj [3] notes, the IT infrastructure is now entangled with digital products and services. Enterprise architecture, here defined to encompass sometimes distinct roles of business architecture, technology/systems architecture, solutions architecture, and application architecture, is tasked with maintaining a longer-term view of the role and management of the entirety of IT infrastructure, even when much of it is provided outside the organization. Historically, this would mean extensive planning and modeling, but these traditional approaches are no longer viable in most organizations as agile development and DevOps move organizations away from occasional releases toward continuous delivery and even, in an increasing number of organizations, deployment.

As the scope of digital business requires expansion beyond individual projects, business units, and often organizations, the role of enterprise architecture becomes more important and the attendant complexity puts more pressure to increase agility just to keep up. Providing the elasticity needed to accommodate rapid changes in scale calls for agility and sourcing beyond the organization. Speed requires agility throughout the entire software supply chain. Finally, novel means of value creation need an architecture capable of supporting innovation in whatever forms are needed.

These IT trends facilitate, drive, and complicate sourcing IT from outside the organization. Particularly difficult are extant outsourcing contracts and relationships that had not anticipated what can be argued are radical changes in architecture and development.

In addition to the market and existing technology drivers pressuring innovation and agility, emerging technologies in Internet of Things (IoT) and artificial intelligence

amplify the demand. IoT adds orders of magnitude more devices, but worse comes without accepted standards, massive security gaps, and the products and interfaces are evolving rapidly. Artificial intelligence-driven systems are enabling automation and previously prohibitively expensive systems and architectures by eliminating the need for human involvement.

4.2 The evolution of agility in the enterprise

Agile began as an approach to software development, popularized by the Agile Manifesto [4]. For organizations struggling to meet the demands of web-enabled systems with waterfall methodologies, agile had great appeal. Its impact on the effectiveness, efficiency, and morale of development groups was widely noted beyond the web development world and adoption began to spread. While many of the best practices emerge from organizations with massive "web-scale" operations, companies of all stripes have been rapidly adopting agile practices. As Figure 4.1 shows, by 2017, a majority of development teams were describing their methodology as agile.

The desire of organizations to be more nimble in the marketplace and able to adapt more smoothly internally has been a focus of management theory from the outset. More recently, the concepts of agility have been spreading across organizations far beyond the application to software development [6,7]. As internal organizations become more agile, they are increasingly constrained by their relationships with outside organizations. To realize their visions of enterprise agile, they expect and require their vendors to be agile and support their efforts. Velocity is the most common metric associated with agile, typically measured by the magnitude of stories delivered per sprint. Agile teams are also concerned with quality, effectiveness, and, increasingly, value [8].

In many organizations, budgeting has not caught up to the movement to agile, and is routinely still done in annual cycles. Agile development, by design, makes long-term estimation more difficult because it adapts over time, making resource needs difficult to project.

4.2.1 Lean enterprise

Many of the tenets of agile and DevOps overlap with the concepts of lean, which grew out of manufacturing, most notably the Toyota Production System. The principles of lean (eliminate waste, building quality, create knowledge, defer commitment, deliver quality, respect people, and optimize the whole) were ported to lean software development [9] and have been applied to companies as a whole [10, 11]. Toyota is of course well known for its management of outside vendors, and the move to lean organizations is similarly focused on the entire value chain.

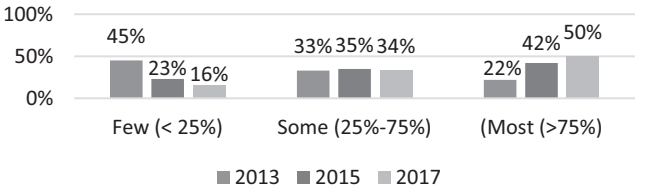

Figure 4.1 Agile adoption trends [5]

4.3 DevOps

Supporting digital business requires agility from all facets of IT work, not just development. As agile development progressed, the desire to implement ran into major impediments when the code was passed to operations. Weeks- and even months-long processes for integration were commonplace in many organizations, particularly larger ones, but made the shift to continuous deployment infeasible. DevOps tries to address some of the greatest constraints on achieving agility. The term "DevOps" emerged in 2009 following a talk at the O'Reilly Velocity 2009 conference [12]. In the intervening years, no standard definition of DevOps has emerged. Most definitions emphasize coordinating efforts of development and operations, focused on the application of agile principles and practices to facilitate more frequent delivery of higher quality systems which deliver greater value to the business. The concept has shown such strong appeal that in less than a decade, according to one survey [13], 78% of organizations have adopted DevOps, and of those, 30% company-wide, with another 28% adopting at the business unit or division level.

Teams delivering code frequently if not continuously, learning from it, and moving forward, is at the core of agile methods. In the early days of agile on websites, once code was ready, it was released immediately to customers. This is still the practice for many websites, but it may not be realistic for many enterprise systems. For instance, retail organizations may lock their systems in the fourth quarter to avoid system issues during their busiest season. Distinguishing delivery of working code from deployment and release of it is essential for enterprises. So finding ways to accommodate frequent/continuous delivery while protecting the integrity of enterprise systems became necessary. Some code may be deployed but not released to customers (similar to the DevOps concept of a feature flag, where code in the production system may be enabled or disabled by setting a simple parameter), while other code may be aggregated for deployment and eventual release [14,15]. This allows the agile developers to maintain momentum and get much needed feedback from clients without disrupting the business.

Much has been written about difficulties in alignment between business and IT, but intense, often overlooked, conflict also exists within IT itself. The development and operations organizations have a long history of tension, with each routinely blaming the other for the too common failure of systems. The conflict between dev and ops is rooted in their varying responsibilities, cultures, and world views. Table 4.1 outlines the stereotypes often attributed by each side to the other.

A major part of DevOps is improving the working relationship between dev and ops. It can be easy to dismiss the value of ops and simply see it as an easily outsourced commodity. The popular term "NoOps" reflects the drive for a state where ops does not require attention at all, and a predictably defensive reaction from ops [17]. But maximizing agility and treating the infrastructure as code will require careful thinking about ops, just as any code does, whether there is any in-house ops staff or not.

4.3.1 IT architecture

The fundamental tenets of service-oriented architecture, that architecture should be modular, well encapsulated, loosely coupled, discoverable, shareable, and distributable, have been widely accepted in theory for decades, going back to the advent of object orientation, but only in recent years, as the pressing need for agility combined with the availability of Cloud and nearly everything as a service (sometimes referred to as XaaS) has widespread

Table 4.1 Stereotypical differences between Dev and ops [16]

Dev's stereotypes of ops	*Ops' stereotypes of Dev*
Ponderously slow to act, whereas Dev feels agile and able to move quickly	Careless and lacks discipline; ops needs to perform heroics to keep the mess running
Wants to maintain the same old obsolete systems that cannot adapt to Dev standards	Likes to play with the latest "toys"
The department of "no" – there's always a reason something can't be done	Thinks anything is possible and there are no costs or other implications to its demands
Runs unreliable infrastructure that negatively affects Dev's credibility	Delivers lousy code that negatively affects ops' credibility
Not creative enough to understand the complex artistry of apps	Doesn't understand the real world, and creates unrealistic test environments
Just a bunch of process zealots and bureaucrats	Prima donnas who practice "art for art's sake"
Cobbles together infrastructure – it's not engineering	A bunch of hackers, not real professionals
Dev can do much better by replacing ops with the Cloud and supporting itself	Ops can do so much better by replacing Dev with packages software and SaaS

implementation of these principles been deemed feasible in many organizations. These attributes are not panaceas. Calibrating the degree of complexity, the size of the modules, and how they connect require sustained effort. This often falls to the enterprise architects, as any single agile initiative may not address these larger issues.

DevOps relies heavily on automation across all areas. Much of this involves self-service, where developers can "create environments, test and deploy code, monitor and display production telemetry" [18]. This self-service may extend to launching code through immutable infrastructure, where the only way code is deployed is through changes in version control, which then re-create the environments such as virtual machines, to prevent variance between production and the assumed code base [19]. Such variance is common – one telecom company found that only 50% of dev and test environments matched production [20] – and the cause of many bugs in production systems. This leads to system images being replaced frequently. At Netflix for instance, the average age of an Amazon Web Services instance is 24 days, and 60% are less than a week old [21]; 50% of Google's code is changed each month [22].

4.3.2 Magnitude of potential impact

Organizations implementing DevOps have reported staggering improvements. DevOps offers greater visibility into processes for the business [23]. The Walt Disney Company credits its DevOps initiative with enabling auto-scaling of its streaming sites, with hundreds of servers created on demand in less than 30 minutes, deployment time on many of its sites dropping from hours to minutes, and an 85% reduction in human errors [24]. The New York Stock Exchange went from 300 to 700 servers per admin and cut provisioning time from one to two days to 21 minutes [25]. While the 700:1 ratio is impressive, it is far less than Facebook's reported 25,000:1 [26]. HP cut development costs by more than 40%, saving over $40 million, and increased its "capacity for innovation" from 5% to 40% [27]. After facing highly publicized struggles with its systems, LinkedIn retooled with DevOps to deploy successfully multiple times daily [28]. Table 4.2 shows the difference in agile metrics between high-, medium-, and low-performing organizations.

Table 4.2 IT performance metrics [29]

Survey questions	*High IT performers*	*Medium IT performers*	*Low IT performers*
Deployment frequency For the primary application or service you work on, how often does your organization deploy code?	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month[a]
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e. how long does it take to go from code commit to code successfully running in production)?	Less than one hour	Between one week and one month	Between one week and one month[a]
Mean time to recover (MTTR) For the primary application of service you work on, how long does it generally take to restore service when a service incident occurs (e.g. unplanned outage, service impairment)?	Less than one hour	Less than one day	Between one day and one week
Change failure rate For the primary application or service you work on, what percentage of changes either results in degraded service or subsequently requires remediation (e.g. leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?	0–15%	0–15%	31–45%

a Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers.

4.3.3 Monitoring and analytics

Historically, monitoring was used to keep track of the technical infrastructure and predict, detect, diagnose, and remediate issues. This was complex enough when dealing with just the hardware and operating systems within a single data center. As systems grew more distributed, monitoring evolved to manage remote locations, often in other organizations. The lower levels of the stack identify network bottlenecks and failure points as well as issues at the hardware and platform levels. While such narrow technical data can be helpful, it does not present a complete picture, and monitoring evolved to move up the stack into middleware, databases, and applications.

The focus on the full stack, from the lowest levels of infrastructure and hardware to the user interface, is an integral part of DevOps, to the point where full stack engineer is now a job title. To be agile, particularly if that involves continuous deployment, all levels of the stack need to be prepared for the changes needed to provide the desired functionality, and to ensure quality of service, the entire stack needs to be monitored.

Because many of the systems were opened up to customers and trading partners, the value of the monitoring data to those outside the information technology organization became apparent. To monitor effectively and provide value to such a range of constituents, organizations have moved from traditional blackbox monitoring, where nothing was known about what happens inside a server, service, or application, to whitebox monitoring, which provides much greater transparency and granularity [30]. Much of this information still comes

from traditional log data, but Turnbull [31] notes that event tracking and metrics often prove more actionable. To facilitate the transparency, instrumentation of applications has become more common, with developers considering monitoring needs when designing and building apps, providing data feeds with a wealth of information. Building monitored applications requires thought in the design of the system to facilitate the subsequent monitoring [32]. As more applications are comprised of components and microservices, it will be even easier to get more data as each component or service will provide its own stream of data for monitoring along with the service.

The data gathered is crucial for the analytics process. The information available goes well beyond traditional network analytics and includes detailed information about applications and services that is useful to developers, ops, and business people across the organization interested in customer and/or user behavior. This commonly includes analysis of user counts, conversion rates, preferences, and behavior.

The skill set for this level of sophistication of analytics is a major change in many organizations, requiring massive data management of petabytes of raw data drawn from a complex array of heterogeneous distributed virtual servers (Etsy tracks over 800,000 metrics [33]), sophisticated understanding of architecture, and advanced knowledge of statistics to generate actionable insights.

4.3.4 Testing and Netflix's simian army

Organizations moving to DevOps need to accept that problems will happen, even if the failure rate may be lower. This comes as a shock in many organizations where failure has historically been chastised if not punished. It also requires changing the nature of service level agreements, or at least our reliance on them. The emphasis is on testing and monitoring and recovering quickly when the inevitable issues arise.

At the more extreme, and for many anxiety-provoking, level is the testing of code on production systems with active users. There are myriad terms for techniques for doing this: A/B testing, where alternative versions of code are deployed simultaneously to separate instances and customers are routed to one or the other based on any business rules that might be helpful, to see which version is more effective; canary rollouts, where code is deployed to subsets of users to see if there are problems; blue/green deployments, where two versions of the production system are maintained, and which one is available to users is alternated as new code is deployed; and dark launches, where code is placed onto production systems but not announced, allowing testing and enhancements until full release to users [34]. Netflix uses these techniques to conduct experiments to optimize the experience of their users [35].

There are three kinds of code: application, infrastructure, and test [36]. Much of software engineering theory evolved while focused on application code, and in many organizations, that is where developer resources are focused. But to achieve agility, a great deal of automation is needed in ops and QA, requiring code be implemented and, often, written.

Netflix has become famous for using such code and testing to limits previously unseen in large organizations. Its Chaos Monkey is designed to turn disable production systems to test claimed resiliency to failure. They later moved beyond the Chaos Monkey to an entire simian army, including Doctor and Janitor monkeys to remove unhealthy and unused instances, a Security Monkey to shut down any instances with violations, a 10–18 monkey for localization and internationalization issues [37]. They even implemented a Chaos Kong which would disable entire Amazon Web Services regions [38]. Many of

the monkeys were made available to the public on GitHub, where Netflix to offer newer versions of services encompassing much of the same functionality (https://github.com/Netflix/SimianArmy).

In addition to being the most robust way to test claimed resiliency, such direct testing of the production environment sends a powerful signal about their willingness to stand behind their claims of reliability and, as long as most tests are passed, it enhances the organization's credibility.

4.4 Agility and DevOps beyond the boundaries of the organization

Distributing agile and DevOps work dramatically increases the complexity of managing it successfully in large part because agile processes are so communication intensive [39]. Many of the agile techniques to enhance communication were developed for co-located groups but many have been adapted to create virtual versions, including Kanban boards, burndown charts, standups, etc. Introducing cross-organizational issues on top of that complexity amplifies the problem.

A myriad of vendors, customers, and other organizations increasingly need to interact to function and support agility. Standards to support this are still evolving, so much of the work must be done more manually. In some cases, outsourced vendors have facilitated this. In other cases, outsourced vendors end up with enormous complexity to manage, essentially playing the familiar role of systems integrator, but in a far more dynamic environment.

4.5 Challenges for enterprises

4.5.1 Technical debt

The pressure to act and implement quickly often generates consequences in the long term. Cunningham [40] referred to the accumulation of such consequences as technical debt. Table 4.3 outlines Kruchten's [41] landscape of technical debt, with the implications for agile and outsourcing.

Table 4.3 Landscape of technical debt [42]

Source of technical debt	*Implications*
Architectural debt	Not considering how choices in elements impact others can reduce quality, impede scaling, and make maintenance slower and more expensive.
Structural debt	Poor design can impede efficiency and maintenance.
Test debt	Insufficient testing can lead to more defects in production and require more maintenance.
Documentation debt	This makes maintenance slower and more expensive.
Low internal quality	This can manifest in external issues, especially performance, and require more maintenance.
Code complexity	This can create inefficiency and increase maintenance expense.
Coding style violations	These make maintenance slower and more expensive and reuse less likely.
Code smells	These surface indications of potential problems with the code often reflect inefficiency, quality issues, and increased maintenance expense.

While taking on such technical debt might be sensible in the same way any debt can provide leverage, finding the time and resources to pay it off is difficult in most organizations as there is competition for the resources. In many cases, the debtor is unaware they have borrowed, particularly if components are outsourced.

The standard advice, obvious and easily given but far more difficult to follow, is to set aside resources to pay down this technical debt. Kim [43] recommends reserving 20% of the budget for non-functional requirements and reducing technical debt. This requires more discipline than many organizations have, particularly given the pressures to be responsive to the business. It also requires knowing what that has been incurred. Rather than responsible financiers regularly making interest and principal payments, many organizations behave more like failed gamblers scrambling to pay the vigorish only when not doing so constitutes an existential threat.

4.5.2 Technological change

Supporting an agile organization delivering continuously is a major technological undertaking, particularly if demands on the systems are also growing. Often all elements of the enterprise architecture need to change regularly. Both Etsy and Google have undergone five "entire rewrite(s) of their architecture from top to bottom" [44].

4.5.3 Organization change

Enhancing agility requires major changes in organizations, often altering power dynamics [45,46]. DevOps also requires rethinking how organizations meet audit and compliance standards. The greatest changes are in separation of duties, formerly handled by having developers hand off code to operations personnel who deploy and release it in production. DevOps achieves compliance through several mechanisms [47]:

- Automation. Deployment and release are handled by a system, rather than a person, which still constitutes separation from the person in development.
- Separate accounts. A temporary account is created for a developer to deploy and release code. This complies with the Payment Card Industry standard which requires separate accounts, but not necessarily separate people doing development and deployment.
- Independent checkpoints. Code is reviewed and signed digitally.

4.5.4 Scaling agile

Agile methodologies were developed for small project that could be completed by a single team. There are limits to the benefits of agile if it is only implemented at the project level in the development organization. The last decade has seen a rapid evolution of adaptations for scaling agile methods to the enterprise level. Finding ways to scale agile to enterprise level has become a cottage industry. Figure 4.2 shows some of the most common approaches in organizations in 2017. Many organizations are also creating hybrids of waterfall and other traditional methodologies and agile. All of the frameworks attempt to use agile principles to coordinate and aggregate work of small agile teams to accomplish larger enterprise goals. While not all of the frameworks describe a formal enterprise architecture role, each acknowledges to a varying extent, the need for a larger, longer-term view.

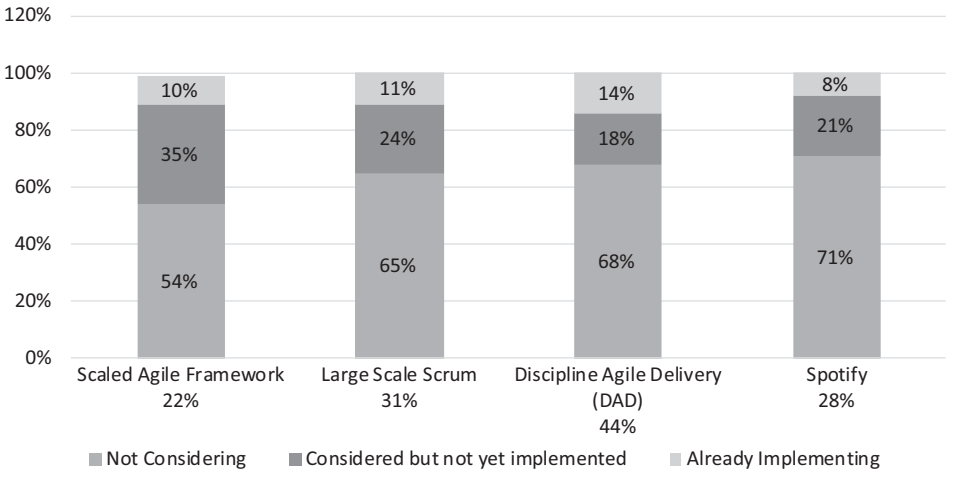

Figure 4.2 Enterprise agile framework adoption [48]

4.5.5 Changing role of EA

Enterprise architects have earned a reputation for being overly plodding; they are often accused of trying to boil the ocean while building models and frameworks. There is now, ironically, tremendous pressure to bring agile/lean principles to the work of enterprise architects. This requires a departure from starting with a massive, all-encompassing design/model, and the changes will be needed at every level of EA. In agile development, the concept of delivering a minimum viable product is a valuable framing to avoid gold plating/waste and delays. Enterprise architecture is working toward developing minimum viable architectures [49], but this quest is far more complex than minimum viable products, in no small part because EA isn't developed as a field, so establishing what constitutes an acceptable minimum is difficult.

The relationship between EA and the organization in many companies must also change. Architects can no longer expect to dictate how systems will be designed, developed, and implemented. Instead, the most effective EA groups are becoming more consultative, providing value just in time to dev and ops.

While agile was initially developed for small systems, at the enterprise level, it is proving helpful to have a conditional approach, where not all systems are treated the same. Two of the main criteria for this are the risk inherent in the systems and the level of innovation required. Gartner distinguishes mode 1 (linear and stable) and mode 2 (nonlinear and uncertain) [50]. Gartner also advocates pace-layering, distinguishing systems of record, innovation, and differentiation [51]. Forrester Research refers to systems of innovation, record, insight, and engagement [52]. The tolerance for risk on these dimensions varies depending on the nature of the system and the perceived need for speed/agility. The greater the granularity of the architecture, the more the gradations these distinctions can have.

4.6 Markets and contracts for agility

A mature or maturing agile organization dramatically complicates the classic "build or buy" dilemma. Forrester Research describes the choices as "build, compose, outsource, or buy" [53]. If the system is well architected, there is a great deal more granularity in

the options. Beyond just the widely known and accepted infrastructure as a service, platforms as a service, and applications/software as a service, the application functionality has been further decomposed into components or services and now into microservices [54] and serverless/Functions as a Service [55].

Systems may employ scores, hundreds, and increasingly thousands of distinct components, many sourced from outside. In addition to traditional vendors, the open-source market is booming. The market for such small elements of code has exploded in recent years.

- There are now more than two million Java unique components in the Central Repository, almost three million unique JavaScript packages in npmjs.org, over 870,000 unique Python components housed in PyPI repository, and over 900,000 .NET components in the NuGet Gallery. There are also more than 900,000 containerized applications housed in Docker Hub [56].

And downloads have similarly grown, as reflected in Figure 4.3.

One survey of widely used repositories found that 5.9% of Java components, 6.2% of JavaScript components, and 3.6% of Python components contained at least one known vulnerability, and the average enterprise downloaded over 125,000 components annually [58]. Perhaps more troubling, only 16% of suppliers actively fix vulnerabilities and for them, the mean time to remediation is 233 days [59].

For many elements, it's a matter of entering a corporate purchasing card number and you are off and running. To further complicate matters, millions of components/services are open source and free for the taking. The governance process for open-source software has proved particularly challenging for organizations, with great risks of the software from both the security and legal perspective. Enforcing this is hard enough in a small development organization, but becomes impossible to do manually when myriad users and external

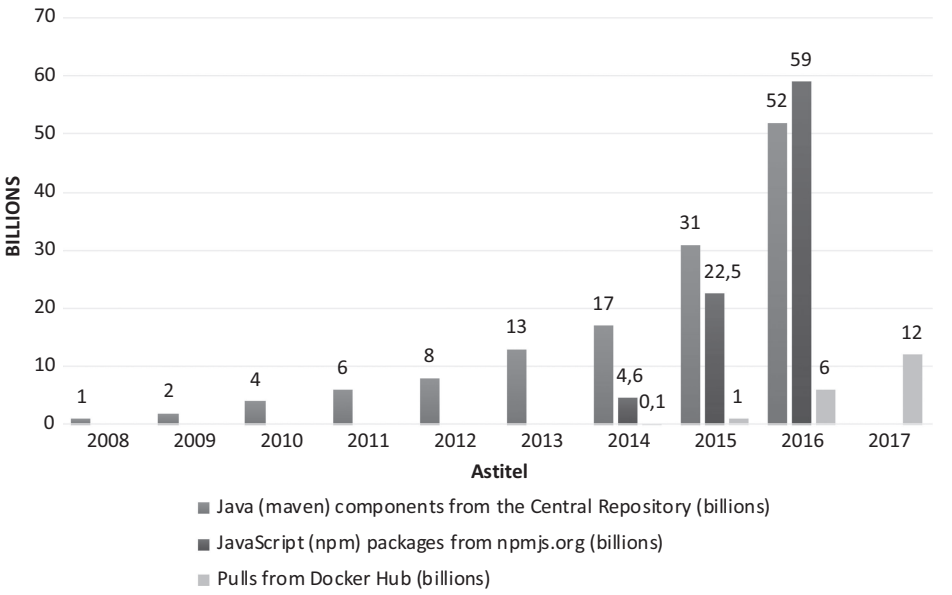

Figure 4.3 Downloads of open-source components [57]

Table 4.4 Percentage of work done manually, by performance [60]

	High performers (%)	*Medium performers (%)*	*Low performers (%)*
Configuration management	28	47	46
Testing	35	51	49
Deployments	26	47	43
Change approval processes	48	67	59

vendors may also be employing open source solutions, whether for complete applications or small components of them. The automated tools, including firewalls to prevent access to the components in the first place, are becoming more sophisticated and are an integral part of any effective governance program. Table 4.4 shows how much more is automated in high performing organizations.

DevOps now moves toward supply chain theory to broaden its scope in an emerging market. But because software is intangible, the degree of distribution and the number of vendors can be orders of magnitude greater than in manufacturing where the theory was formed and has already moved past manufacturing theory in terms of the speed and extent of automation throughout the chain.

Proper tools to support DevOps are critical. The toolset includes local development environments, version control, artifact repositories, testing, infrastructure automation, virtualization, and monitoring tools [61]. These tools are maturing rapidly and are routinely incorporating best practices. Cloud vendors are also evolving, offering much of the same functionality.

The use of automated code generation, sometimes referred to as low code, increases the speed of development and can be used by the user base producing their own code. When these tools first emerged decades earlier, they were often seen as not worth the trouble, as the resulting code was often buggy and inefficient. These objections are fading quickly in large part, not only because the tools themselves have improved dramatically, but also because processing power can offset some of the efficiency issues and time to market is often more important than efficiency or the highest quality levels.

Existing outsourcing relationships often struggle to adapt to the required level of agility. To support agility, many aspects of the traditional sourcing processes must change. The arms-length relationship with the client so common among outsourcers goes against the tenets of agile. But even organizations with close client relationships must still address challenges of distributed agile work, as well as educating clients about their role in agile processes. The contracts are often not amenable to agility and require contortions or renegotiation. Many outsourced firms are not staffed or organized for agile, although as agile matures, there are more new university graduates and experienced professionals with robust backgrounds in agile and even DevOps.

If the outsourced vendor has the entire lifecycle within its operation, they may better positioned for agile and DevOps than their clients. They have greater economies of scale in the monitoring and much of the work to be done can be held to standards set and audited within the organization [62]. Few organizations have the scale and skills to do all of their DevOps work in-house economically [63]. Thus, it is likely the vendor is still managing a complex software supply chain that extends beyond its boundaries, but they may still be in a better position to negotiate better deals with vendors and identify issues at one client that can be immediately addressed at others. But they may also be forced to collaborate with competitors, creating complex issues around coopetition and potentially encumbered by contract terms.

4.6.1 Sourcing strategy

Kleim [64], cited by Santy [65], offers seven steps involved in sourcing strategy:

- Determine the business case for or against outsourcing.
- Search for vendors.
- Select a vendor.
- Conduct negotiations.
- Consummate an agreement.
- Manage the agreement.
- Determine the business case to decide whether to renew, renegotiate, or terminate a contract.

All of the phases with the possible exceptions of managing the agreement and determining whether to renew face tremendous pressure to move more quickly. Complicating matters, many purchasing departments are not set up for agility and in fact are well designed as impediments to it. Creating formal business cases to justify purchasing outside code or services takes time both to create and to review. The expectation of pre-approving vendors with a rigorous process that can take weeks or months to complete creates delays in adding innovative vendors. Cloud, emerging technologies, microservices, and serverless are driving companies toward smaller vendors. The approval process may be too expensive to be worth doing for such small vendors, who may not be approved anyway because of stringent viability criteria emphasizing scale and track record of vendors. The search process itself has become more burdensome as the variety of vendors increases exponentially. Once viable vendors are found, many organizations require the detailed process for selecting vendors that may include several stages, including formal requests for proposal and perhaps trials of the products and services. Reaching an agreement may require multiple layers of authorizations, building further delay into the process.

4.6.2 Cloud

In addition to readily available infrastructure as a service to replace in-house data centers (or those housed at outsourcers), much of the code that once needed to be written in-house or outsourced is now readily available as Cloud services. And once a vendor is chosen, the contracts for Cloud are in many ways better. They are certainly easy, requiring only a brief form and a corporate purchasing number. The clickwrap agreements are standard, which reduces consolidation complexity if they need to be aggregated. But there are dramatic limits to customization and negotiation and most buyers don't even glance at the terms, which typically heavily favor the vendor.

Many Cloud vendors will not offer, perhaps because they cannot, full transparency into their systems. This is often a consequence of how they configure multi-tenancy to reduce their own costs. This introduces limits to what can be monitored, and to what extent.

4.7 Consulting

Organizations routinely look for help in making the move to agile. Every major consultancy purports to offer such services, from strategic advice and consulting to instrumental help and extra hands. Myriad boutique firms also compete in the space. But contracting for agile

is more complex, particularly if clients want to move beyond time and materials billing. Some client organizations manage some of the risks and complexity of pricing by employing business outcome-driven contracts where the vendor is paid, at least in part, based on the results the organization achieves with the system. Such value-based pricing is becoming a more common component in contracts, but few vendors are interested in having it represent a major portion of their revenue stream.

Traditional full outsourcing is certainly still an option, but it is increasingly rare that a chosen external vendor would have all of the work and systems in-house. Given the rapidity of developments, particularly related to the IoTs, multisourcing has become a requirement for most enterprises. But the drive for integration and analytics as well as end-to-end service management has led to a need for the vendors to coordinate. The ecosystem has also grown to include customers. The increase in agility has been highly correlated with an increased focus on and involvement of customers.

4.8 Conclusion

Agility, driving both EA and DevOps, does not change the longstanding need to be open to outsourcing and to choose how and when to do so carefully. While it certainly requires detailed thought and heavy engagement of the client, lack of client engagement was often a complaint of outsourcing organizations, and there is long history of information systems research about improving the relationship between the two. Agility drives greater value to clients, which could help dramatically in bringing vendors and clients together.

References

1. Bharadwaj, A., El Sawy, O. A., Pavlou, P. A., & Venkatraman, N. V.: Digital Business Strategy: Toward a Next Generation of Insights. *MIS Quarterly*, 37(2), 471–482 (2013).
2. Nielsen, T.: *Maverick Research: CIOs Should Forget Digital Business and Concentrate on Core IT*. Gartner, Stamford, CT, October 7 (2016).
3. Bharadwaj, A., El Sawy, O. A., Pavlou, P. A., & Venkatraman, N. V.: Digital Business Strategy: Toward a Next Generation of Insights. *MIS Quarterly*, 37(2), 471–482, p. 472 (2013).
4. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Jeffries, R.: Agile Manifesto. (2001). http://www.agilemanifesto.org
5. Lo Giudice, D.: *The State of Agile 2017*. Forrester Research, Inc., Cambridge, MA, December 14 (2017).
6. Rigby, D. K., Sutherland, J., & Takeuchi, H.: Embracing Agile. *Harvard Business Review*, 94(5), 40–50 (2016).
7. Humble, J., Molesky, J., & 'O'Reilly, B.: *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media, Inc., Sebastopol, CA (2014).
8. Swanton, B., & Norton, D.: *Use the Right Metrics in the Right Way for Enterprise Agile Delivery*. Gartner, Stamford, CT, November 2 (2017).
9. Poppendieck, M., & Poppendieck, T.: *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley, Boston, MA (2003).
10. Humble, J., Molesky, J., & O'Reilly, B.: *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media, Inc., Sebastopol, CA (2014).
11. Ries, E.: *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Books, New York, NY (2011).
12. Sharma, S. (2017). *The DevOps Adoption Playbook* (Kindle ed.). Wiley IBM Press, Indianapolis, IN, Kindle location 743 (2017).
13. RightScale: RightScale 2017 State of the Cloud Report. p. 20 (2017). http://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf
14. Sharma, S. (2017). *The DevOps Adoption Playbook* (Kindle ed.). Wiley IBM Press, Indianapolis, IN, Kindle location 1069 (2017).

15. Humble, J., Molesky, J., & O'Reilly, B.: *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media, Inc., Sebastopol, CA, Kindle location 3430 (2014).
16. DeMartine, A., & Bittner, K.: The Seven Habits of Highly Effective DevOps. Forrester Research, Inc., Cambridge, MA, October 2, p. 3 (2014).
17. Humble, J., Molesky, J., & O'Reilly, B.: Lean Enterprise: How High Performance Organizations Innovate at Scale. O'Reilly Media, Inc., Sebastopol, CA, Kindle location 5523 (2014).
18. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 461 (2016).
19. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 2517 (2016).
20. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (First ed.). IT Revolution, Portland, OR (2016).
21. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 2588 (2016).
22. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 2648 (2016).
23. Sharma, S. (2017). *The DevOps Adoption Playbook* (Kindle ed.). Wiley IBM Press, Indianapolis, IN, Kindle location 902 (2017).
24. Earnshaw, A.: Disney's DevOps Journey: A DevOps Enterprise Summit Reprise. https://puppet.com/blog/disney-s-devops-journey-a-devops-enterprise-summit-reprise, February 24 (2015).
25. Puppet: NYSE and ICE: Compliance, DevOps and Efficient Growth with Puppet Enterprise. Retrieved from https://www.puppet.com (2015).
26. Frazier, M.: Facebook Won't Hire You For Its Data Center. *Bloomberg Businessweek*, September 25 (2017).
27. Gruver, G., & Mouser, T.: *Leading the Transformation: Applying Agile and DevOps Principles at Scale*. IT Revolution, Portland, OR, Kindle location 106 (2015).
28. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 1805 (2016).
29. Forsgren, N., Humble, J., Kim, G., Brown, A., & Kersten, N.: State of DevOps Report. https://www.puppet.com, p. 23. The 2017 DevOps Survey is the property of Puppet, Inc. and DevOps Research and Assessment, LLC. All rights reserved (2017).
30. Turnbull, J.: *The Art of Monitoring* (Kindle ed.): James Turnbull & Turnbull Press, New York, Kindle location 465 (2014).
31. Turnbull, J.: *The Art of Monitoring* (Kindle ed.): James Turnbull & Turnbull Press, New York, Kindle location 483 (2014).
32. Turnbull, J.: *The Art of Monitoring* (Kindle ed.): James Turnbull & Turnbull Press, New York, Kindle location 5399 (2014).
33. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 3758 (2016).
34. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 656 (2016).
35. Govind, N.: A/B Testing and Beyond: Improving the Netflix Streaming Experience with Experimentation and Data Science. *Netflix Technology Blog*. 13 July (2017). https://medium.com/netflix-techblog/a-b-testing-and-beyond-improving-the-netflix-streaming-experience-with-experimentation-and-data-5b0ae9295bdf
36. IT Revolution: *An Unlikely Union: DevOps and Audit: Information Security and Compliance Practices*. IT Revolution, Portland, OR (2015).
37. Izrailevsky, Y., & Tseitlin, A.: The Netflix Simian Army. *Netflix Tech Blog*. July 18 (2011). https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116

38. Basiri, A., Hochstein, L., Thosar, A., & Rosenthal, C.: Chaos Engineering Upgraded. *Netflix Technology Blog*. September 24 (2015). https://medium.com/netflix-techblog/chaos-engineering-upgraded-878d341f15fa
39. Diel, E., Marczak, S., & Cruzes, D. S.: Communication Challenges and Strategies in Distributed DevOps. Paper presented at the 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE) (2016).
40. Cunningham, W.: The WyCash Portfolio Management System. *ACM SIGPLAN OOPS Messenger*, 4(2), 29–30 (1993).
41. Kruchten, P., Nord, R. L., & Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, 29(6), 18–21 (2012).
42. Kruchten, P., Nord, R. L., & Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, 29(6), 18–21 (2012).
43. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 1178 (2016).
44. Kim, G., Debois, P., Willis, J., & Humble, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, Portland, OR, Kindle location 3534 (2016).
45. Gruver, G., & Mouser, T.: *Leading the Transformation: Applying Agile and DevOps Principles at Scale*. IT Revolution, Portland, OR (2015).
46. Stoneham, J., Thrasher, P., Potts, T., Mickman, H., DeArdo, C., & Limoncelli, T. A.: *DevOps Case Studies: The Journey to Positive Business Outcomes*. IT Revolution, Portland OR (2016).
47. IT Revolution: *An Unlikely Union: DevOps and Audit: Information Security and Compliance Practices*. IT Revolution, Portland OR (2015).
48. West, M., & Norton, D.: *Market Guide for Enterprise Agile Frameworks*. Gartner, Stamford, CT, July 17, p. 4 (2017).
49. Erder, M., & Pureur, P.: *Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World* (Kindle ed.). Morgan Kaufmann, Waltham, MA (2016).
50. Norton, D.: *Bimodal in an Agile-Everywhere World*. Gartner, Stamford, CT, September 30 (2016).
51. West, M., & Norton, D.: *Market Guide for Enterprise Agile Frameworks*. Gartner, Stamford, CT, July 17 (2017).
52. Bartoletti, D.: *Take the Wheel: Build Your Cloud Computing Strategic Plan Now*. Forrester Research, Inc., Cambridge, MA, July 12 (2017).
53. Lo Giudice, D., & Condo, C.: *Master DevOps for Faster Delivery of Software Innovation*. Forrester Research, Inc., Cambridge, MA, November 21, 2017.
54. Balalaie, A., Heydarnoori, A., & Jamshidi, P.: Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3), 42–52 (2016).
55. Hausenblas, M.: *Serverless Ops*. O'Reilly Media, Sebastopol, CA (2017).
56. Sonatype: 2017 State of the Software Supply Chain, p. 13. Retrieved from www.sonatype.com (2017).
57. Sonatype.: 2017 State of the Software Supply Chain, pp. 15–16. (2017). www.sonatype.com
58. Sonatype.: 2017 State of the Software Supply Chain, p. 17. (2017). www.sonatype.com
59. Sonatype.: 2017 State of the Software Supply Chain, p. 14. (2017). www.sonatype.com
60. Forsgren, N., Humble, J., Kim, G., Brown, A., & Kersten, N.: State of DevOps Report. https://www.puppet.com, p. 27. The 2017 DevOps Survey is the property of Puppet, Inc. and DevOps Research and Assessment, LLC. All rights reserved (2017).
61. Davis, J., & Daniels, K.: *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*. O'Reilly Media, Inc., Sebastopol, CA, Kindle location 3935 (2016).
62. Sharma, S. (2017). *The DevOps Adoption Playbook* (Kindle ed.). Wiley IBM Press, Indianapolis, IN, Kindle location 6853 (2017).
63. Sharma, S. (2017). *The DevOps Adoption Playbook* (Kindle ed.). Wiley IBM Press, Indianapolis, IN, Kindle location 6909 (2017).
64. Kliem, R.: Managing the Risks of Outsourcing Agreements. *Information System Management*, 16(3), 91–93 (1999).
65. Santy, S., & Sikkel, K.: Sourcing Lifecycle for Software as a Service (SAAS). Paper presented at the EPJ Web of Conferences (2014).