The Path to DevOps

Erik Dörnenburg, ThoughtWorks

// IT in businesses is now entirely a team activity. While we still need experts with deep technical knowledge, we must focus on how to get people from all disciplines working together effectively. //

THE ROLE OF IT in the business world has changed dramatically over the past decades. New technologies and techniques allow enterprises to get much more out of IT, while at the same time increasingly sophisticated business models have pushed IT to investigate and deliver novel solutions. These may be based on new technology; however, larger breakthroughs have required both new technologies and a rethinking of how people are organized to make use of the technology.

Agile software development has led the way, and now the DevOps and DesignOps movements are hitting the mainstream. In my opinion, IT in businesses is now entirely a team activity. While there is, as ever, a need for experts with deep technical knowledge, we must focus our attention on how we get people from all disciplines working together effectively.

Technology in a Supporting Role

In the early days of IT in business, the 1960s, IT played mostly a supporting role. IT systems were usually introduced to make existing processes faster or to reduce the cost of executing a given process. The focus was on the bottom line, and the goal was to improve operational efficiency.

A major shift came with the PC. It was more powerful than green-screen terminals, and it found its way onto more desks. As a result, new areas of the business were supported by IT, including areas for which no off-the-shelf software existed. So, in response, enterprises started asking their internal IT departments to build custom applications.

There was obviously some collaboration between the people commissioning and using the IT systems, often simply referred to as "the business," and the people building the systems, the IT department. However, IT was still considered a tool, and IT departments were treated as a cost center. This led organizations to a project-centric view of IT and to introducing a clear separation between development and operations. The project-centric approach created a culture in which the on-time and on-budget delivery of a fixed set of requirements within a project context became the supreme goal. IT departments created ever more formal and elaborate processes to prove that they held up their end of the bargain. I have seen more than a few cases where that was deemed more important than creating actual value for the organization as a whole.

The PC brought another innovation: office packages, including word processors and spreadsheets. Through macros, embedded programming capabilities, and integration mechanisms, these packages became a powerful platform that the business side used to implement complex solutions on its own. One of the key drivers for this was that the IT departments had come up with the rigid processes mentioned above, effectively discouraging the business side from interacting with them, when, actually, the business needed closer collaboration.

Building systems outside the IT department became so common that a term was coined for it: shadow IT. Unfortunately, while improving

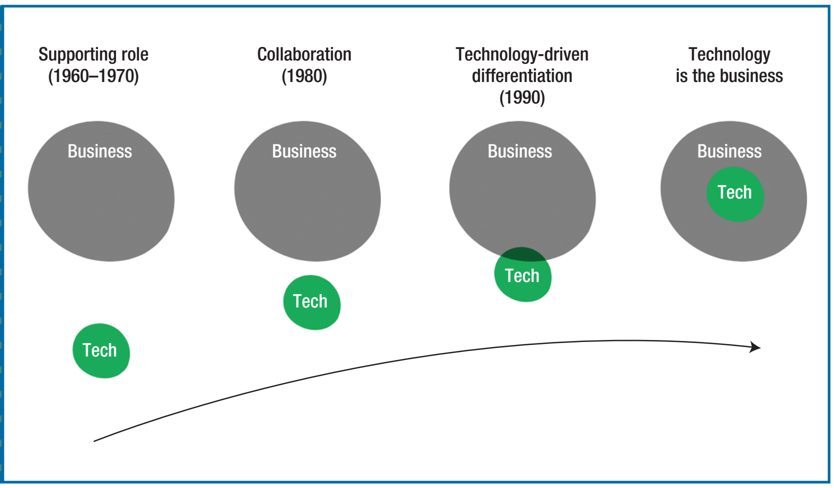
Supporting role (1960–1970)
Collaboration (1980)
Technology-driven differentiation (1990)
Technology is the business
Business
Tech

FIGURE 1. The changing role of technology. For many companies, technology has become the business.

collaboration, shadow IT is also expensive. Studies have found that 30% to 50% of IT spending can be on shadow IT, and it introduces a significant business risk.[1]

Technology Becomes the Business

Closer collaboration gave some businesspeople an important insight: while they could treat IT as a cost center, they could, instead, also view IT as an enabler. When used well, technology could give their organization a competitive advantage in the market, not because they could deliver existing products more efficiently but because they could offer something new. They had hit on the concept of technology-driven differentiation. They became less interested in figuring out how to reduce the cost of delivering an IT project and more interested in finding out how much money they could make with a new technology-led product.

With the rise of the dot-com economy, this trend continued, and companies sprang up for whom technology was the business. They did not differentiate their products through the use of technology; they existed only because of technology (see Figure 1).

For companies that adopted a view of IT as an enabler, the then-dominant project-focused view of building software did not work well.[2] Some organizations experimented with different approaches, culminating in the Manifesto for Agile Software Development in 2001.

From personal experience, I can say that one industry that saw the new potential of IT early was investment banking. In trading especially, technology became a key differentiator. As a response, many front-office groups adopted agile software development in the early 2000s.

Another industry that visibly used a collaborative and product-focused approach was the start-ups of the dot-com boom. Certainly, the dot-com crash, as well as stories about haphazard engineering and operations practices, did much to discredit the working mode of these start-ups, However, in hindsight, it seems obvious that they had been onto something.

Cycle Time Is Key

When an organization relies on technology to differentiate itself or when technology is at the core of its business, cycle time becomes key. The quicker an organization can take a new business idea and turn it into software running in production, the better.

Previously, organizations had focused mostly on two factors: on reliability, in the sense of a reliable process to turn business requirements into working software, and on throughput, that is, delivering the maximum number of features with a given development capacity. Analogous to the saying, "Never underestimate the bandwidth of a station wagon loaded with tape," high throughput is not linked to a short cycle time. In fact, a lot of organizations had improved, or attempted to improve, reliability and throughput at the expense of cycle time.

Conversely, organizations who focus on cycle time cannot simply ignore reliability and throughput, which means that they have to come up with an approach that delivers all three. Close collaboration lies at the heart of the solution.

Agile software development promotes collaboration and introduces short feedback cycles on multiple levels. By releasing a version of the software that is still incomplete but already usable, real user or customer feedback becomes available. This feedback is then used to direct the development effort, which increases the reliability of the process to deliver what is needed. Throughput is increased, too, because little to no effort is put into unneeded features. This concept is often described as "waste minimization," a term borrowed from lean manufacturing.[3]

Businesses that have technology at their core often do not know with certainty what they need to deliver next in order to stay ahead of their competition. Product managers merely have hypotheses that need validation. In this situation, the effect of fast feedback is even more pronounced. Release cycles common in project-centric organizations, measured in quarters or even years, would be prohibitively long.

More than Agile Software Development

Following many implementations of agile development approaches, around 2008, it became apparent that there were shortcomings on both ends of the software development process that still had to be addressed in order to reduce cycle time. On one end, having well-tested, continuously integrated software in version control does not help the business. The software has to be in production, and the path to production must be short and free of obstacles, which it usually was not. On the other end, business people often cannot describe the requirements for the systems they need to the development teams, whether they are agile or not.

From the Internet start-ups, some of which have evolved into giants, we learned to shift from project-focused to product-focused thinking.[4] Rather than separating the lifecycle of software into distinct phases, with separate teams for each, the team boundaries are redrawn, bundling responsibility for continued evolution of software and its operation in one team. Werner Vogels, chief technology officer of Amazon, famously summarized this approach as “you build it, you run it.”[5] These cross-functional teams are normally kept small, around 8 to 12 people, so that they can maintain a focus on a specific aspect of the overall product. Spotify was one of the early companies to implement these concepts, also exploring strategies to scale them.[6]

While this approach was relatively easy to follow in start-ups, larger enterprises struggled to transition to this new world. It had been difficult to learn agile development. Replacing teams grouped by functional specializations such as databases, front ends, or back ends with small, cross-functional teams caused further friction and resistance. And bringing together development and operations was exceptionally hard because in many cases they were explicitly separated, to the point where different vendors, sometimes located on different continents, could be responsible for either.

DevOps and DesignOps

At the same time, practitioners from development and operations teams felt increasing levels of pain and frustration as their priorities collided more and more often. Development teams had been taught to focus on on-time and on-budget delivery. They often did this at the expense of stability and maintainability, because they knew they would not be measured on the latter, never mind being held accountable. Operations teams, on the other hand, were measured on stability and the cost of operations. This made them uncomfortable with any new release and led them to introduce complicated formal processes, resulting in less collaboration and long cycle times.

Similarly to the early stages of agile software development, it was practitioners who joined in a grass-roots movement, called DevOps, to resolve the conflict. The devopsdays conference series, which began in 2009, played a central role. Reports show that while the problem was well understood, practical solutions were not commonplace.[7] Nonetheless, practitioners began to explore how they could, in their organizations, transition to a model that not only allowed but also promoted close collaboration between development and operations teams.

DevOps and, more specifically, the collection of techniques and tools known as continuous delivery cleared the path to production. In the best case, individual chunks of functionality, captured by user stories, can be developed and deployed into production in day or two, not in weeks or months. Today, sizeable IT departments, comprising many product teams, can manage hundreds of releases of related pieces of software into production every week.

This possibility of dramatically reduced cycle time motivated many organizations to try to move toward a DevOps culture. For decision makers who still needed convincing, the *2014 State of DevOps Report* provided evidence that “high IT performance correlates with strong business performance.”[8] The authors later provided more insights into why they felt confident making this assertion.[9]

With cycle time, we should measure the time between an idea and its realization as software in production. User stories in agile development are already much more than an idea, and many organizations that introduced agile development struggled with writing appropriate user stories. Oftentimes, large backlogs of detailed stories were created, bearing an eerie resemblance to detailed specification documents of days past. In response, the focus turned to improving where the user stories came from.[10]

In a movement almost symmetrical to DevOps, organizations started to include user experience into their IT departments, to ensure that the actual needs of the business were represented in the user stories. The user experience experts brought new techniques such as design thinking and user research that added further feedback cycles. They also pushed for explicitly capturing hypotheses about features and validating them with data gathered from actual use of the system, a technique known as hypothesis-driven development.[11]

Now, quickly picking up insights from the DevOps movement, designers and developers are taking steps to collaborate closely. They use new tools that enable them to work jointly on the same technical artifacts. This is sometimes known as DesignOps and is practiced at Airbnb, for example.[12]

Enabling Techniques and Technologies

While this article focuses on the organizational changes that allowed businesses to get more out of IT, it is important to note that a number of breakthroughs in technology and software engineering techniques went hand in hand with the organizational changes.

The practices collected in Extreme Programming, most notably test-driven development and continuous integration, gave development teams the necessary boost in confidence to release software more frequently. Because of the absence of unpredictable merges, these practices also made the process more reliable.

Service-oriented architecture, later refined by the concept of microservices, provides independent evolvability. With such an architecture, the small, cross-functional teams described above can each work independently on their part of the overall IT landscape. Techniques such as consumer-driven-contract testing[13] and micro frontends[14] often allow teams to release services into production without testing them in an integration environment first. In fact, for organizations that release software many times a day, it is simply not possible to run extensive integration tests in a preproduction environment.

Another important innovation was virtualized infrastructure. Rather than manually deploying an application on an OS manually installed on a physical server, modern services are running in virtual machines or containers, and each step of their deployment is fully automated. In the past, the lifecycle of an OS installed on a server was longer than that of the applications running on it. New versions of an application would be installed onto existing servers, often sharing the same OS with other applications. Today, the lifecycle of the entire (virtual) machine is linked to the lifecycle of a single version of a service, and a machine usually exists with only one service deployed on it. This greatly reduces dependencies and improves the reliability of deployments.

When developers and operations people started working closely together, the developers brought more automation with them. Rather than configuring servers manually, they saw infrastructure as code.[15] Scripts that can set up the entire deployment infrastructure, including software-defined networking, are managed just like the source code of the services running on them. Rebuilding the infrastructure is treated in the same way as releasing a new version of a service, and it can be done with the same speed and reliability.

When technology becomes the business, or when a business relies on technology to offer differentiated products, collaboration is key. We have seen a shift to small, cross-functional teams where engineers with different specializations collaborate. The DevOps and DesignOps movements bring closer collaboration between all roles involved in the life of an IT solution. In combination with agile software development, this has allowed the business and IT to collaborate efficiently.

Today, organizations that have mastered the concepts described in this article consider their IT landscape a digital platform. Business-centric services that can evolve quickly and independently, combined with frequent and reliable releases, finally put the old dream of reusable and recombinable components within reach for them. The organizations can innovate and move fast because their whole approach to IT allows them to experiment at scale.

References

1. P. Bendor-Samuel, "How to Eliminate Enterprise Shadow IT," *CIO*, 11 Apr. 2017; https://www.cio.com/article/3188726/it-industry/how-to-eliminate-enterprise-shadow-it.html.
2. S. Narayan, *Agile IT Organization Design: For Digital Transformation and Continuous Delivery*, Addison-Wesley, 2015, ch. 8 and 9.
3. M. Poppendieck and T. Poppendieck, *Implementing Lean Software*

Development: From Concept to Cash, Addison-Wesley, 2006, ch. 4.
4. D. West et al., *Product-Centric Development Is a Hot New Trend*, Forrester, 2009.
5. J. Gray, "A Conversation with Werner Vogels," *ACM Queue*, vol. 4, no. 4, 2006, pp. 14–22.
6. H. Kniberg and A. Ivarsson, "Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds," blog, 14 Nov. 2012; https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify.
7. D. Ackerson, "Back to the Roots: Bridging the Deployment Gap," blog, 3 Nov. 2009; https://www.agileweboperations.com/devopsdays-2009.
8. N. Forsgren Velasquez et al., *2014 State of DevOps Report*, Puppet, 2014; https://puppet.com/resources/whitepaper/2014-state-devops-report.
9. N. Forsgren, J. Humble, and G. Kim, *Accelerate: Building and Scaling High Performing Technology Organizations*, IT Revolution, 2018.
10. L. Ratcliffe and M. McNeill, *Agile Experience Design: A Digital Designer's Guide to Agile, Lean, and Continuous*, New Riders, 2011.
11. S. Klepper and B. Bruegge, "Impact of Hypothesis-Driven Development on Effectiveness, Quality, and Efficiency in Innovation Projects," *Lecture Notes in Informatics*, Gesellschaft für Informatik, 2018, pp. 181–188.
12. A. Cleave, "DesignOps at Airbnb: How We Manage Effective Design at Scale," Airbnb; https://airbnb.design/designops-airbnb.
13. I. Robinson, "Consumer-Driven Contracts: A Service Evolution Pattern," 12 June 2006; https://www.martinfowler.com/articles/consumerDrivenContracts.html.
14. T. Söderlund, "Micro Frontends—a Microservice Approach to Front-End Web Development," *Medium*, 6 July 2017; 14. https://medium.com/@tomsoderlund/micro-frontends-a-microservice-approach-to-front-end-web-development-f325ebdadc16.
15. K. Morris, *Infrastructure as Code: Managing Servers in the Cloud*, O'Reilly Media, 2016.

ABOUT THE AUTHOR

ERIK DÖRNENBURG is a software developer, a consultant, and a Head of Technology at ThoughtWorks, where he helps clients write custom software. Over the years he has worked with numerous technologies and technology platforms, aiming to understand their potential for solving real-world problems. Dörnenburg received a degree in informatics from the University of Dortmund. Contact him at erik@thoughtworks.com.