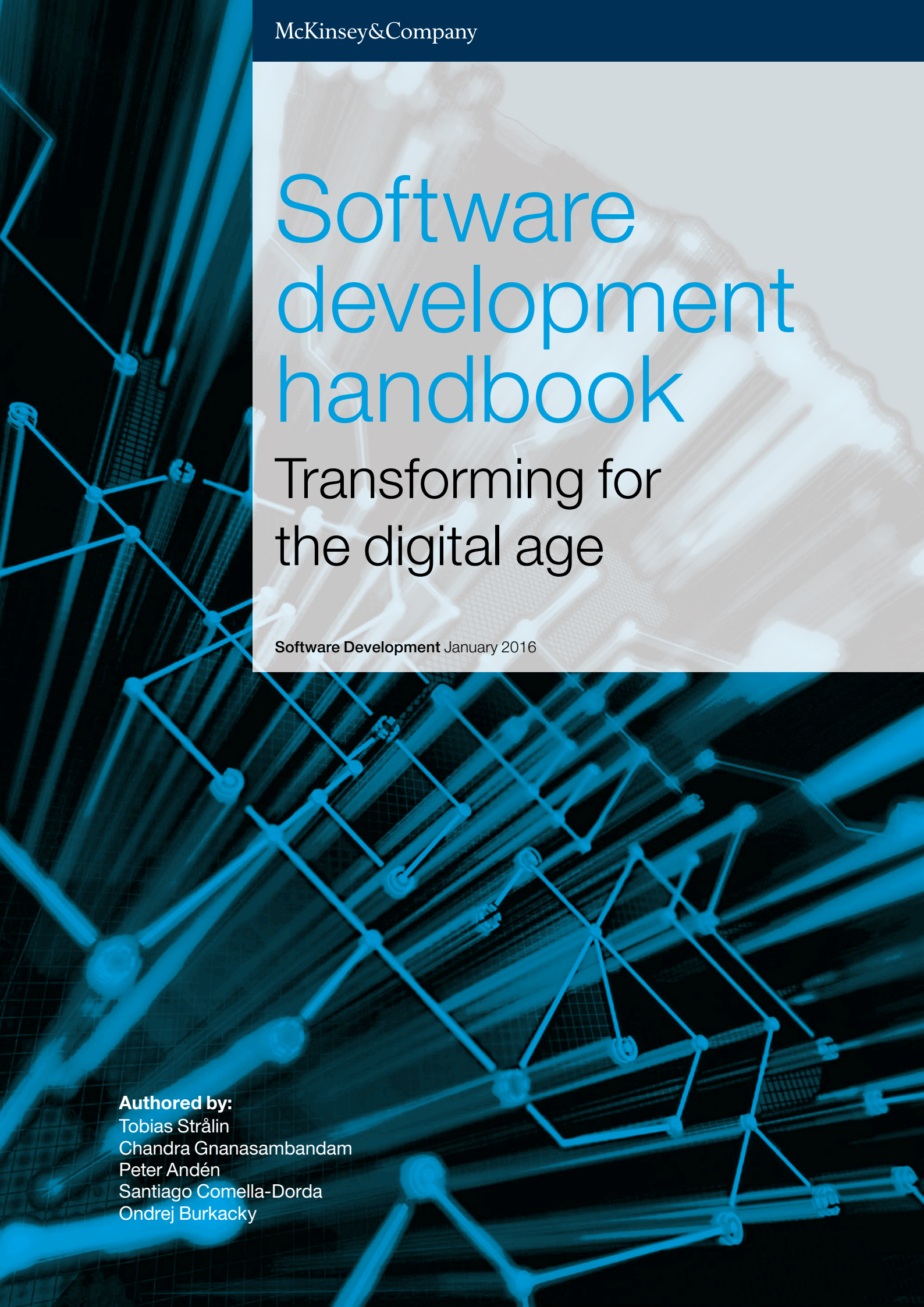
McKinsey&Company

# Software development handbook

## Transforming for the digital age

**Software Development** January 2016



**Authored by:**
Tobias Strålin
Chandra Gnanasambandam
Peter Andén
Santiago Comella-Dorda
Ondrej Burkacky

# Software development handbook

## Transforming for the digital age

**Software Development** January 2016

**Authored by:**
Tobias Strålin
Chandra Gnanasambandam
Peter Andén
Santiago Comella-Dorda
Ondrej Burkacky

# Preface

Software development is no longer the domain of Silicon Valley tech start-ups. It is increasingly essential to companies big and small, across all industries, and around the world. Organizations that once dedicated nearly all of their resources to hardware are rebalancing their priorities. Producers of medical technology products, semiconductors, and automobiles are all faced with the reality that quality software—thus, superior software development—is as essential to their success as excellence in sales is.

Despite this, McKinsey research has revealed a sizable gap between top and bottom performers. This gap in performance means that top companies can accelerate the flow of new products and applications at much lower cost and with markedly fewer glitches than other companies. The articles in this software development handbook offer a path forward for organizations looking to position themselves on the winning end of that distribution.

Some of the articles have been commissioned specifically for this handbook, while others have already been published. All of them offer leaders insights into the various elements of building and sustaining successful software development organizations—including benchmarking, organizational design, and development efficiency—and help leaders get the answers that will inform strategy development.

We hope that leaders will find this handbook valuable and that it will help them steer their organizations into a new software future.

Finally, we would like to thank all who have contributed to this document and have shed light on the shadowy elements of software development.

Tobias Strålin

Chandra Gnanasambandam

Peter Andén

Santiago Comella-Dorda

Ondrej Burkacky

# Contents

# The perils of ignoring software development

Peter Andén, Chandra Gnanasambandam, Tobias Strålin

Software can no longer be ignored. By assessing the external stakes and internal capacities, CEOs can build a software organization that enhances their products and differentiates them within the market.

As digital technologies relentlessly reshape competition, products and services increasingly depend on software for differentiation and performance. Software is behind smartphones and other interfaces that guide consumer interactions; algorithms orchestrate productivity-boosting process automation; wearable devices loaded with software monitor the health and performance of athletes and patients alike. Despite the mission-critical nature of software, it gets surprisingly little attention in the C-suite. Most often, it is relegated to functional managers, several levels down the organization, who manage teams of programmers.

Research suggests, however, that companies pay a price when they undervalue the strategic importance of developing quality software. During 2013, McKinsey examined three core metrics for software development performance at more than 1,300 companies of varying sizes and across all regions of the world. We found not only stunning differences between the highest- and lowest-performing organizations but also sizable differences between the top and average performers. Top-quartile companies developed software upwards of three times more productively than companies in the bottom quartile. They had 80 percent fewer residual design defects in their software output. Our research also showed that the companies benefited from a 70 percent shorter time to market for new software products and features. This performance gap means that top companies can speed up the flow of new products and applications at much lower cost and with markedly fewer glitches than other companies can.

## The coming revolution

Such performance leverage will become even more important as the transition from hardware- to software-enabled products accelerates. Today's shift resembles what occurred in the 1970s, when digital electronics began replacing the mechanical and analog technologies that underlay products from calculators to TV sets. The number of top 100 product and service companies that are software dependent has doubled, to nearly 40 percent, over the last 20 years. Value is shifting rapidly as hardware features are increasingly commoditized and software differentiates high- from low-end products. And ever more miniaturized computing power means that the value of embedded software in products is expected to keep growing.

Already, software enables an estimated 80 percent of automobile innovation, from entertainment to crash-avoidance systems, according to automotive software expert Manfred Broy (an electric vehicle may have 10 million lines of code, and a typical high-end car can have many times that).[1] Interfaces will become even more sophisticated—and critical—as a growing variety of products, from home appliances to mobile medical devices, are designed around smart screens. As software-enabled customer interactions become the rule, revenues from digitized products and channels are expected to exceed 40 percent in industries such as insurance, retailing, and logistics. The software-led automation of manufacturing and services has generated rising output while reducing costs. And companies with consistently high-performing software experience less operational downtime and develop products with

[1] *Robert N. Charette, "This car runs on code,"* IEEE Spectrum, *February 1, 2009, spectrum.ieee.org. See also* Digits, *"Chart: A car has more lines of code than Vista," blog entry by Brian R. Fitzgerald,* Wall Street Journal, *November 11, 2013, blogs.wsj.com.*

fewer glitches that mar the consumer experience. In a letter to shareholders, General Electric CEO Jeffrey R. Immelt offered a view of where things are headed: "We believe that every industrial company will become a software company."[2]

## Raising the profile of software development

CEOs need to determine whether they have the right organization and capabilities to compete in an environment where software continues to change the game. Asking three questions can help start the process:

What are the strategic stakes? CEOs and their top teams should quickly get up to speed on how software could be differentiating or disrupting their current businesses and industries. Scania creates a competitive edge for its trucks through advanced software features that give drivers real-time information on how to optimize fuel use and maximize safety. Semiconductor maker MediaTek invested in software-based reference designs[3] in the wireless chips it produces for smartphone manufacturers. The new offerings upended competition in the high-volume, low-end smartphone industry, leading to a tenfold increase in MediaTek's sales of wireless chips within a single year, as customers benefited from lower development costs, faster times to market, and increased design flexibility.

Where does our software power reside? Outside the technology sector, senior software leaders are rarely in the top-management hierarchy. Many companies manage software strategy three to five levels down in the organization, within scattered departments often dedicated to designing and building hardware platforms. Siloed software expertise makes it difficult to assemble a strategic core of software leaders who can think cross-functionally about innovation or productivity.

One path forward is to give a software development executive a seat at the top management table. Companies can do so by establishing an office—chief of software development—that reports to the CEO, much as companies have done in recent years with the role of chief digital officer or chief information security officer. Such an executive is well positioned to help high-ranking executives understand how the software development performance of their company stacks up against that of its peers. This software development leader can also communicate the risks of substandard processes and the strategic importance of improving software development performance by overhauling organizational structures, development methods, and metrics.[4]

How do we build the software development muscle needed? In many industries (again, apart from high tech), hardware and mechanical engineers dominate the engineering leadership, so it is difficult to attract the talent needed for cutting-edge software R&D teams. Companies can break through in two ways. The first is mounting an effort to change the organization, developer by developer: building a software powerhouse organically, from existing internal organizations, while targeting top software companies to get strong contributors who will become software champions and talent magnets. A second option is acquiring a software company to break into new technology areas and get a higher level of software capability. Walmart followed this approach, acquiring a number of smaller start-ups to strengthen its position in e-commerce as well as social and mobile retailing.

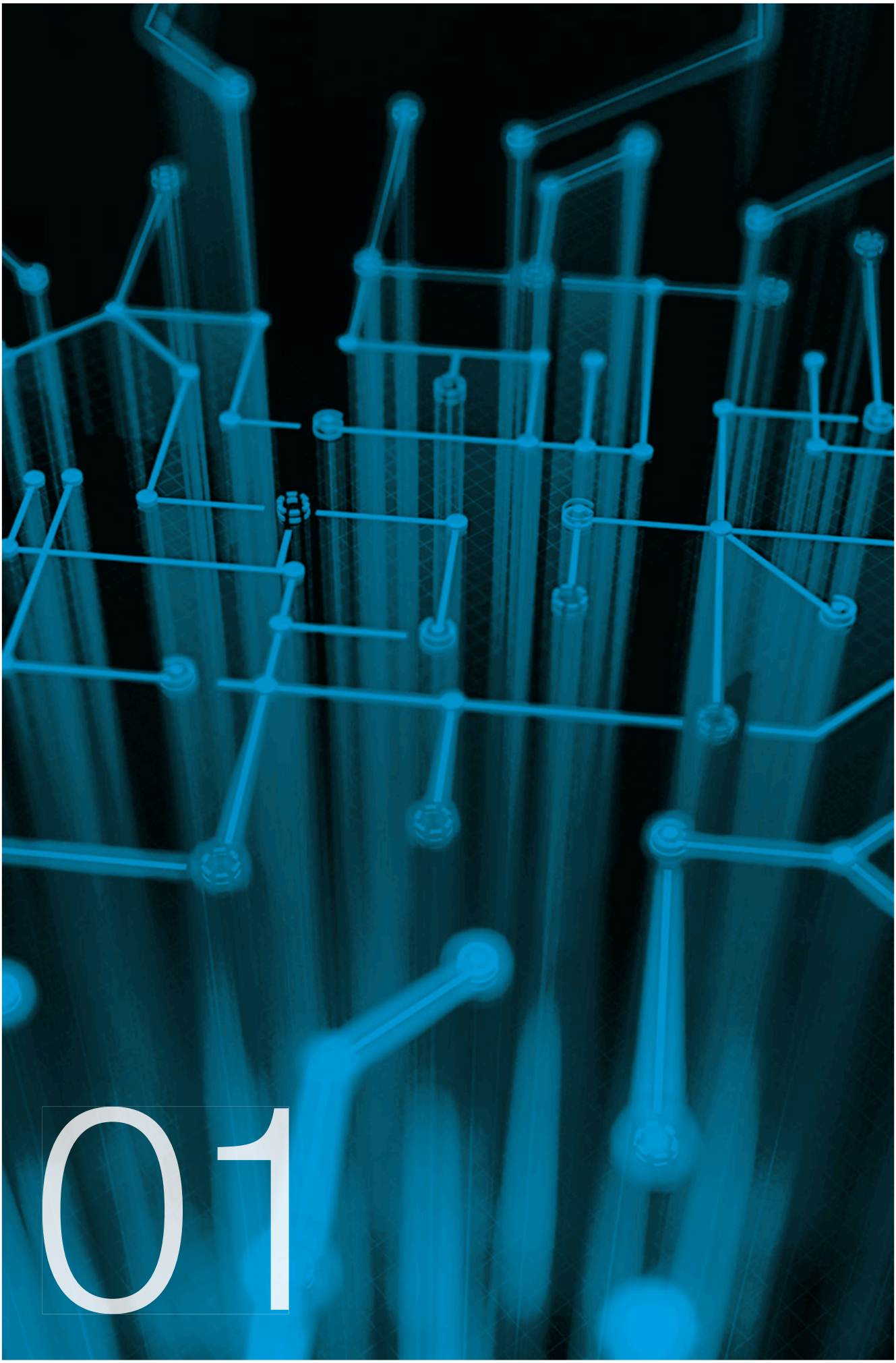
In either approach, companies need to follow through with software-friendly operating models that incorporate agile working methods, flexible hours, and motivational tactics (such as internal competitions) that spur developers to engage with innovative and challenging projects.

[2] *Jeffrey R. Immelt, "Letter to shareowners,"* 2013 GE Annual Report, *ge.com.*
[3] *A technical architecture for a system that can speed up customized software development.*
[4] *Colocating diverse software design teams in the same facility and using analytics to predict quality levels are ways top companies are getting more leverage from advanced design methods and setting ambitious but realistic goals for teams.*

Unconventional hiring processes (coding contests or testing online gaming skills, for example) may be needed to screen candidates and identify top talent—as some top digital players already do. There's no escaping the competitiveness of today's software talent marketplace, which is particularly challenging for large companies seeking to build their capabilities. As digital technologies continue to reshape markets, though, there's little alternative. Embracing the rising strategic importance of software, and viewing its development as a crucial competitive battlefield, are keys to success for an ever-growing number of companies.

*The authors wish to thank Karim Doulaki, Simone Ferraresi, and Shannon Johnston for their contributions to this introduction.*

*This piece was first published in* McKinsey Quarterly.



# 01

# Software or nowhere: The next big challenge for hardware OEMs

Peter Andén, Ondrej Burkacky, Jörn Kupferschmidt, André Rocha

Digitization opens up a world of possibilities. Value chains and business processes are being redrawn in every industry. Software lies at the heart of this disruption. Leaders need to formulate clear software strategies in order to unlock its full potential and retain their competitive advantage.

It's hard to name an industry that isn't feeling the change resulting from the increasing importance of software. It is having a major impact on product development, customer service, and organizational development across industries. Even in traditional industries that haven't seen productivity increases in decades, software innovations hold the promise of transformation and growth. Take the case of the construction industry. Imagine every construction site digitally designed and all plans checked using virtual reality. Once construction begins, buildings are built with automated laser precision using preassembled objects that know where they belong and how they should be connected.

For medical device manufacturers, software is already a major innovation driver and regarded as a critical functionality in more than 80 percent of devices. Container terminals for ocean shipping are becoming fully automated, with all physical operations being orchestrated by a central terminal operating system. Service operations across industries are being transformed by the ability to monitor and predict imminent failure. And in industries such as media and telecommunications, software is having major impact on product development, customer service, and organizational development.

The Internet of Things revolution, cheap connectivity, and advanced analytics are part of a growing digital landscape predicted to have an $11 trillion global impact. This revolution will ultimately change all industries, and software lies at the heart of it. For the leaders of many companies, however, the approach to software in the digital landscape may not be clear. The challenge for executives is one of distilling the particular opportunity for their organizations from this large and amorphous value. They will need an approach that addresses the business potential along with an organization and capabilities that unlock the value of that potential.

## Business strategy: Identifying software's value

A company's software business strategy will outline the products and services it should offer. Asking themselves the following questions will help leaders determine how and where digital fits and how software can support it: Where is the value? Which fields beyond our core offering should we seek to address? Do we need to self-disrupt?

McKinsey's research suggests that three elements are key for the success of companies in more traditional industries transforming to the digital realm:

Go granular. Before an executive can set forth an organization's software strategy, he or she must build an understanding of the value chain and the customer. Knowing customer pain points and sources of inefficiency sheds light on which problems the company may be able to solve. Analyzing the moves competitors have taken in the recent past will provide clues as to how they believe the future will look. Leaders will need to consider all elements of the current delivery system, including the opportunity to establish new business models (see text box "Software's disruption gives rise to new business models," page 13). A granular exploration of the value chain and customer needs is critical in understanding the evolving opportunity and competition landscape, particularly in the B2B realm.

Build on strengths. Established companies typically do not have a start-up's agility, and competing with start-ups on their turf will likely increase the risk of failure. Instead, leaders in established businesses should think about how to use the assets they already have to their advantage. Such assets may include a strong customer base, customer trust, a broad hardware portfolio, and domain knowledge. It can also be long-standing and powerful partnerships with suppliers, IT companies, connectivity providers, and/or competitors. These strengths may give established businesses the upper hand when it comes to scaling quickly and efficiently as well as controlling various value points—specifically the customer-related ones.

Create structural advantage. Beyond customer-centric value exists the opportunity for established businesses to create structural advantages. Executives need to evaluate which of today's strengths they can turn into structural advantages tomorrow. Do they have assets that allow them to be a driving force in an ecosystem, creating a development and distribution platform? Companies may offer access to their strengths in return for shaping how others innovate in the ecosystem to deliver a superior customer journey from multiple, independent products. Similarly, executives may be able to use their influence today to contribute directly to setting industry standards. Or they may be able to craft partnerships beyond the reach of smaller start-ups, creating de facto standards for their industry. Using assets unique to established companies gives these businesses an advantage in controlling the structures of the new ecosystem. Leaders can assess their organizations' capacity to use these assets to influence the development of the value chain and the related software ecosystem.

## Organization and capabilities: Unlocking software's potential

Organizations with a traditional hardware focus are accustomed to product life cycles lasting years. Software, however, enables lightning-fast product development and retooling—and the accompanying software strategy must take the life cycle from years down to months or even weeks. Once leaders have identified the value potential and gained a perspective on how their companies fit in (or can shape) the new ecosystem, it is then time to create the organization and capabilities that will allow them to capture this value.

McKinsey's work in this area reveals three enablers to help established businesses get where they need to be.

Think end to end. Leaders need to take a holistic approach when setting up their software businesses. It all starts with defining what software solutions they need to deliver—in particular, identifying the most appropriate technology platforms for each case and designing a modular architecture that ensures scalability and consistency. It then moves into the full set of capabilities—in terms of both customer and development life cycles—and organizational model required to run a high-performing software business. Customers need to be understood better, involved earlier, and engaged with in a much more continuous dialogue. Organizations must now—more than ever before—be willing to engage with and learn from their customers. The much faster prototyping for software—for example, with mock-ups and digital design labs—means that software should be developed with customer input from early on in the process. Given the faster evolution in the market, these skills are paramount. Finally, companies need to take a careful look at their software road maps and evaluate whether they are planning for the right level of investment.

Focus on talent. Software and design talent is a defining, rare resource. Obtaining talent is hard, particularly if your company is a nontraditional software employer. Attracting talent for key positions is key in order to use these leaders to hire further digital talent, and as a sign that it is a good decision to work for this company. Pulling

## Software's disruption gives rise to new business models

Software is shaking up businesses around the world and across industries. The opportunities introduced aren't limited to start-ups or the tech industry. New business models are being made possible, and established companies from automotive to construction should consider how they can take advantage of them.

**As-a-service models** charge customers by usage or on a subscription basis. These new payment models turn the income that used to be generated by "one-off" sales into recurring revenues.

**Software development and distribution platforms** are digital spaces that enable third-party developers to innovate, create, and sell software using some of the platform owner's assets—for example, their customer base or data.

**Intellectual property rights (IPR)** models deliver recurring revenues, for example, from licensing fees for data standards. IPR models also create space for add-on services on top of the primary product, such as best-usage consulting.

**Data-driven models** monetize crowd-sourced information as opposed to selling particular products or services. Monetization may happen directly or indirectly through the pricing or customization that micro-segmentation allows.

resources together in a consolidated software unit, and elevating it in the organizational chart, will help convey the message that software is strategically important to the company. If the company's location, for example, is a barrier to attracting talent, opening a second location may be a worthwhile investment.

Embrace cultural differences. Software talent may come from a different background than a company's traditional employees. Companies may need to look beyond the sources to which they have become accustomed to find software talent. Traditional, hardware-focused organizations may find that differences in work styles and/or expectations exist between their traditional employees and their growing software organization. It is crucial that the leadership embraces the new talent and understands that any tension that arises is just a necessary growing pain.

The journey from a traditional, hardware-focused company to one with successful software offerings—either stand-alone or in products—is long and arduous. It is important that companies embark on this journey while software is still a small share of their product portfolio. Companies need to do the heavy lifting now. Otherwise, they risk missing out or even becoming obsolete in tomorrow's digital landscape.


02

# Software status: Diagnosing development performance

Peter Andén, Ondrej Burkacky, Tobias Strålin

Companies aspiring toward better software development end games must first understand their starting points. Beginning any software development improvement initiative with a comprehensive diagnosis gives companies the advantage of rooting their improvement blueprints in a deep understanding of their current positions.

Software is an increasingly important component of organizational success across sectors. The successful development and delivery of products and services hinges more and more on a company's ability to effectively develop quality software. The differences that successful approaches to software development can make are clear. Just looking at companies' software development units alone, outcomes between top and bottom performers vary drastically. The top quartile of units is more than three times as productive as the bottom quartile, achieves five times the development throughput, and has six times fewer design defects (Exhibit 1).

Behind the performance of the top companies is their ability to thoughtfully address three questions that underlie the fundamental drivers of software development success:

What software is being developed? Companies need to assess how they prioritize different feature requirements and how they then scope the work and manage requirements (including late requirements). They should also assess how they set up the software architecture and system design to drive efficiency, for example, maximizing code reuse and ensuring a modular software architecture with clear interfaces and few interdependencies.

How is the software developed? Process is the name of the game here. At the outset, project planning and efficient resource management need to be evaluated. Companies must also assess their current software methodology and process to see if there are new and better processes available to increase productivity, time to market, and quality—for example, by moving away from traditional waterfall methods to agile software teams and development teams with integrated operations expertise (DevOps).

Where is software developed? The actual location of the development is also important. Companies must look specifically at their decisions to outsource versus develop internally and the inner workings of the in-house parts of the organization focused on software development. One additional area to look into is how many sites are currently working with the same code base. McKinsey research shows that every site added to a software project results in a productivity loss of 15 percent.

Climbing out of software development mediocrity requires careful analysis. The path toward software development improvement is a highly tailored endeavor, as no two organizations require the exact same approach. McKinsey has developed a five-week diagnostic that helps organizations understand their current performance. Diagnosis of a company's software development function typically comprises three phases: benchmarking output performance, assessing root causes, and identifying key improvement initiatives.

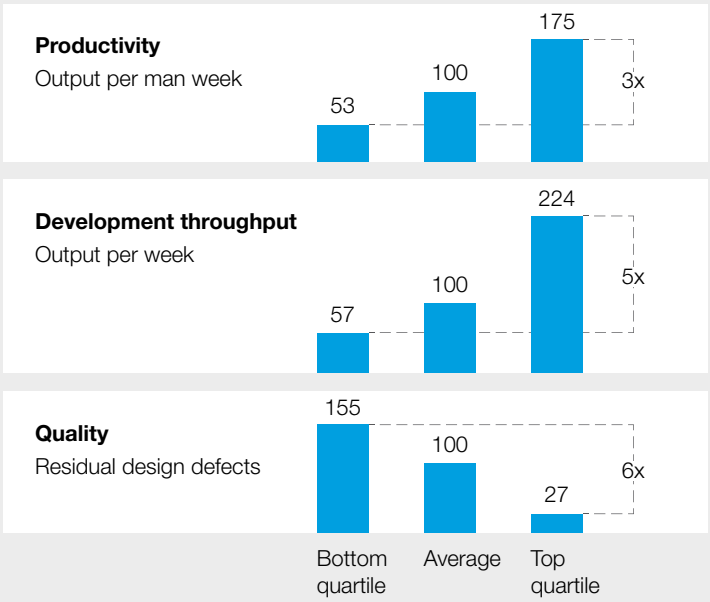
## Benchmarking output performance

Giving companies a clear sense of how they stack up against their peers globally is the first component of the software development diagnostic. Key to meaningful benchmarking is the ability to compare a company's performance

**Exhibit 01**

**Top organizations significantly outperform in all aspects of software development**

Average indexed to 100




Source: Numetrics-embedded software project (a McKinsey Solution), October 2013 (n > 1,300)

to a large set of development projects that not only spans the globe but also represents significant diversity of company size and software development methodology. The diagnostic phase will clearly situate a company's performance in the areas of productivity, time to market, and quality within the global context.

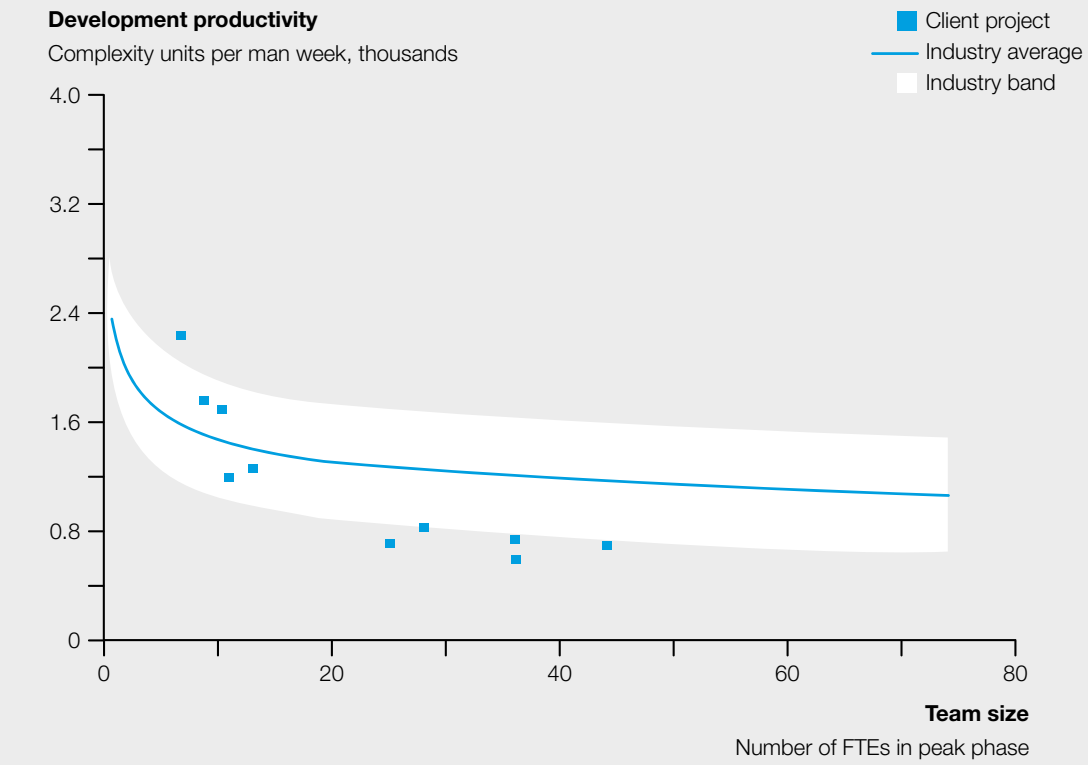
Benchmarking productivity relies on an analysis of a company's design complexity. That is, how does the effort that a company spends on software development compare with the normalized effort required? An analytics engine takes several data points across the complexity dimensions of a company's software requirements, architecture, coding, testing, and hardware and calibrates them against information from other industry software projects.

These complexity calculations—measured in units per man week—can then be graphed along the industry average (Exhibit 2). The complexity rating enables benchmarking across time, team, and software releases.

Benchmarking time to market requires first determining a company's time overrun (i.e., delays in project delivery). Then that overrun percentage is normalized against industry averages. The diagnostic takes delay benchmarking to another level by determining the schedule risk of a company's ongoing projects. By looking internally at where finished products land at the intersection of development productivity and team size, a schedule baseline can be created. New projects get plotted on the curve, and outliers become quickly identifiable as schedule risks.

**Exhibit 02**

**Productivity is benchmarked by looking at project complexity versus team size**




Source: McKinsey analysis

Benchmarking quality looks at the defect-to-project size ratio and gives companies information about how the quality of their software design compares to the industry average. Instances where a project of a particular size has significantly more defects than the average project of that size may require root cause analysis.

## Assessing root causes

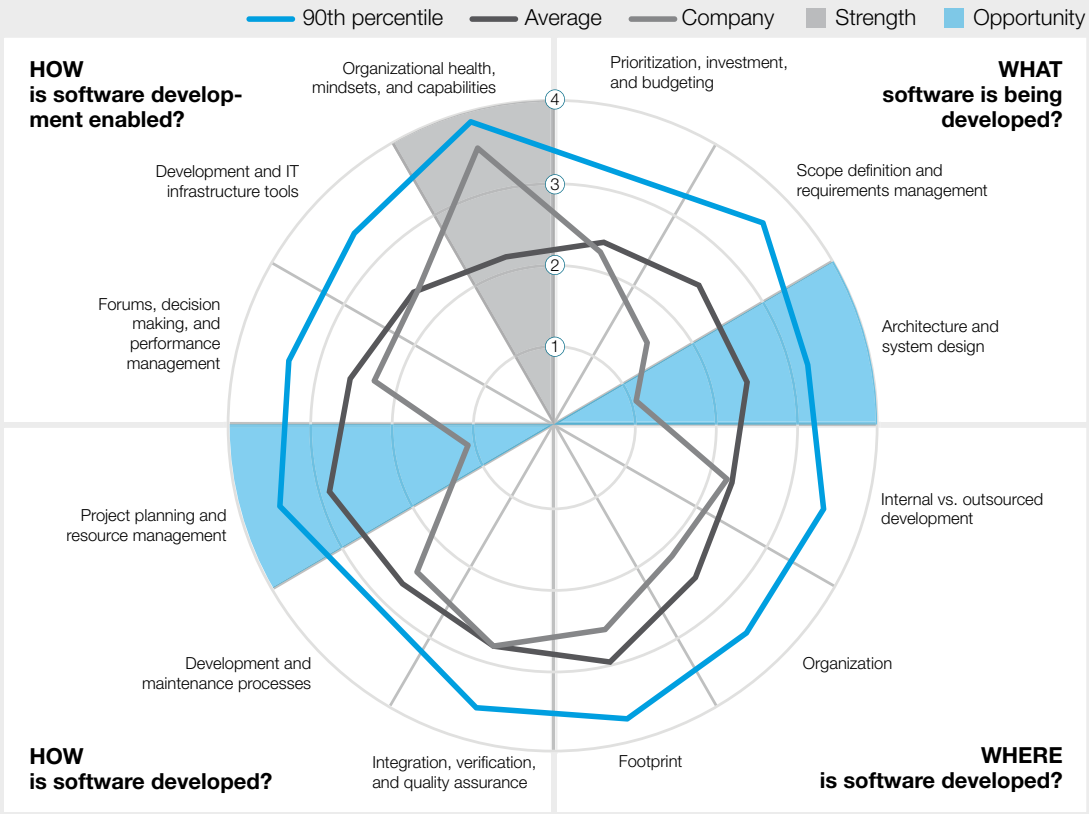
After establishing exactly how a company compares to its peers—and, in some cases, to itself—specific tools can be implemented to discover what is behind the performance.

SD fingerprint. The software development "fingerprint" gives an initial overview of a company's strengths in more than 20 activities across the four major drivers of software development (Exhibit 3): What software is being developed? Where is the software developed? How is software developed? How is development enabled? Understanding a company's capabilities in these activities and then plotting them against industry averages is the first insight into which improvement initiatives will help the company achieve greater software development success.

Site competence heat map. Another tool in the diagnostic toolbox identifies the efficiency of a company's software development footprint, i.e., the competence of each of the sites within the organization that play a role in software development. The site competence heat map breaks down an organization's business units by software development competence and regional

**Exhibit 03**

**The embedded software development fingerprint maps a company's capabilities against industry averages, revealing strengths and opportunities**




Source: MIT, McKinsey analysis

location. The heat map reveals fragmentation and, thus, excessive complexity in projects. This tool offers early insight into the potential for a company to consolidate its software development footprint.

Value stream mapping. Software development is a process, and value stream mapping takes an end-to-end perspective to uncover pain points and bottlenecks along the chain. The various software development activities are broken down by organizational actor and then mapped along the development process. Input is solicited from cross-functional team members regarding what works well and what does not. The result is a concrete overview of the "where," "what", "when," and "who" of the challenges to be addressed and initial thoughts on how to address them.

Overall process efficiency (OPE). OPE analysis takes a team-by-team look at the time spent per team member on software-development-related activities. These activities include total processing time, meetings, and even breaks. By comparing how much time a company spends on a given activity to industry averages, this analysis can identify value-creating versus wasteful efforts.

## Identifying key initiatives

The five-week diagnostic gives sufficient insight for an early, initiative-based action plan. Companies that have undergone the diagnostic have been able to define and plan for quarter-by-quarter software development improvement initiatives over one to two years. These initiatives have included immediate plans to pilot project management

processes and longer-term initiatives to blueprint customer feedback processes. Regardless of which organization-specific improvement levers are selected in this phase of the diagnostic, the chances are high that some sort of capability-building initiative will be among them.

In addition to defining the initiatives, the diagnostic also offers specific projections of the short- and long-term impact of those initiatives. The improvement potential of the various initiatives in the areas of productivity, time to market, and quality is quantified and also translated into bottom-line impact. For one company, the diagnostic led to the definition of an improvement initiative with the potential to increase productivity by 22 to 31 percent, reduce time to market by 10 to 25 percent, and improve software quality by reducing defects by 20 to 45 percent. These improvements translated into a possible revenue increase of $100 million and potential cost reduction on the order of $50 million.

Software development aptitude is growing in importance for companies seeking excellence in customer service and a real competitive advantage. Improvements begin with a diagnostic that benchmarks a company's performance against its peers globally and continues with an assessment of the success drivers and pain points. The diagnostic also points a company in the direction it needs to go to improve and highlight potential bottom-line impact.


03

# Teaching elephants to dance (part 1): Empowering giants with agile development

Peter Andén, Santiago Comella-Dorda, André Rocha, Tobias Strålin

Getting agile development right and at scale requires new processes, governance models, capabilities, and mindsets.

Good software is hard to build. The history of software development is full of projects that took too long, cost too much, or failed to perform as expected. Agile software development methods emphasize tight collaboration between developers and their customers, rapid prototyping, and continuous testing and review (see text box "Agile development," page 24) and have evolved as a direct response to these issues.

Fans of the agile approach say that it improves many aspects of their software development processes, such as an increased ability to handle changing customer priorities, better developer productivity, higher quality, reduced risk, and improved team morale. In an effort to quantify these benefits, we made use of the McKinsey Numetrics database.[1] This proprietary benchmark contains data on the approach, costs, and outcomes of more than 1,300 software projects of different sizes, from different industries and using different programming languages. When we compared the cost, schedule compliance, and quality performance of the 500 or so projects that used agile methods with those that applied the "waterfall" methodology, the agile projects demonstrated 27 percent higher productivity, 30 percent less schedule slip, and three times fewer residual defects at launch (Exhibit 1).

## Old habits die hard

For many companies, however, the move to agile development is a significant cultural shift. Not all organizations succeed in the transition. In one survey of software development organizations, almost a quarter of respondents said that the majority of their agile projects had been unsuccessful.[2] Asked to pinpoint the root cause of their problems, respondents most often cited a corporate culture that was not compatible with agile methods or a lack of experience in the use of those methods.

When we talk to senior executives about the potential benefits and risks of adopting the agile approach, three questions commonly arise: How can we modify the agile approach to work in a large, complex organization like ours? Can we apply agile techniques across all layers of the software stack or just in end-user applications? How do we roll out the agile approach across our organization? Let's look at each in turn.

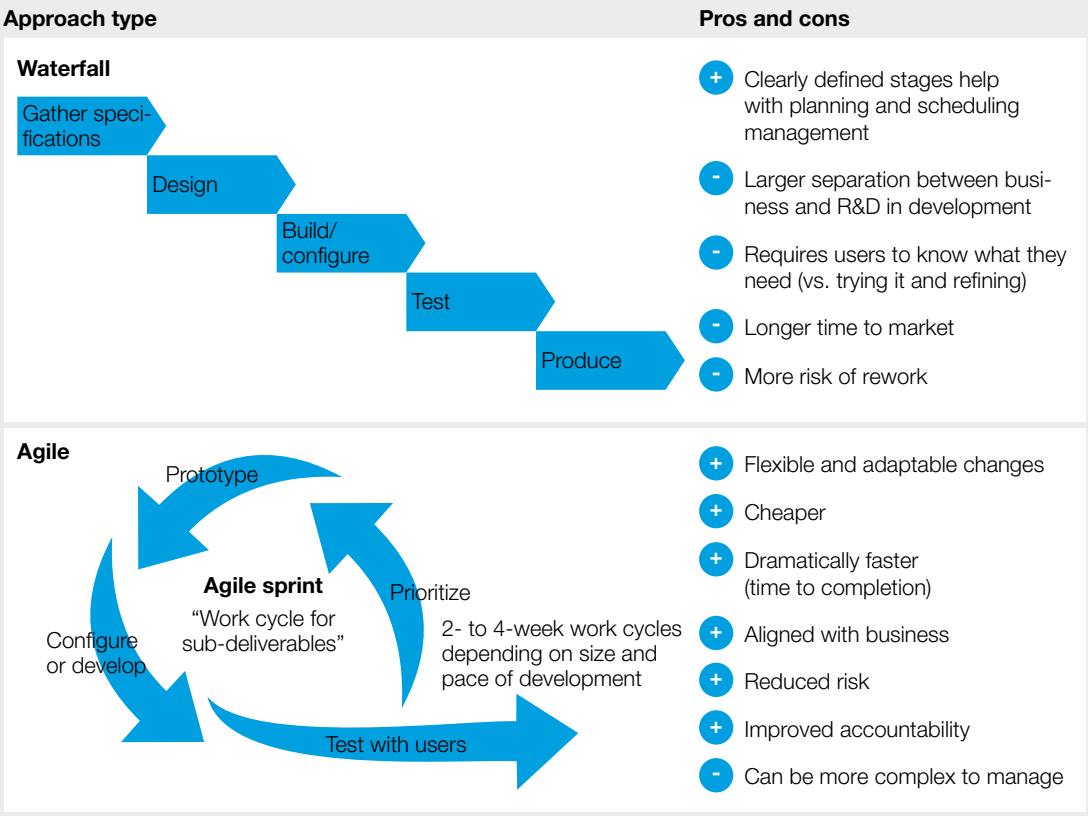
## Making agile work in large projects and large organizations

The agile approach is compelling in its simplicity: one team reporting to one product owner conducts one project. To maintain the key benefits of flexibility, clear communication, and close collaboration, the best agile teams are small: usually five to ten members. Translating that structure to large, enterprise-level software projects with many tens or hundreds of developers can be tricky. As development teams get bigger, they quickly become unwieldy as progress slows, communication gets difficult, and the benefits of agility evaporate.

[1] *See more at http://www.mckinsey.com/client_service/semiconductors/tools_and_solutions*
[2] *Source: "State of Agile development" 2010 survey by VersionOne*

**Exhibit 01**

**Agile software development has a number of advantages over conventional methods**




Source: McKinsey analysis

A more effective approach is to maintain the small size and working characteristics of the core agile teams, and to adopt an architecture, organizational approach, and coordination process that shield those teams from additional complexity. The first step in this approach is building robust modular software architecture with clear interfaces and dependencies between modules. To this end, many organizations choose to have a team of architects moving ahead of—and laying the groundwork for—their development teams. Often, however, organizations have to deal with a large legacy code base, making modularizing every technical component challenging. In these situations, leading organizations are adopting a two-speed architecture, so that certain elements of the stack are modularized to enable agile developments, while legacy elements are encapsulated with a relatively stable application programming interface layer.

With these foundations in place, individual teams can then work on their own part of the product, with multiple teams reporting to a common product owner (PO). The PO is responsible for managing the "backlog" of features, work packages, and change requests and for coordinating the release of product versions. A separate integration team will help the PO make optimal short-term planning decisions as customer requirements and dependencies change. Development work takes place in two- to four-week "sprints" during which the agile teams operate with a high degree of independence. Between sprints, a joint product demo, sprint retrospective, and planning stage ensure all teams maintain coordination.

Regardless of the size of the organization or the project, one tenet of agile development remains true: it is good for all developers to work at a single location. The benefits from improved communication and ease of collaboration that arise from close physical proximity are hard to overstate. While some organizations do manage to run effective distributed development teams, analysis of our software project database reveals that organizations pay an average productivity penalty of 15 percent for each additional development site they use. The change from one site to two incurs the largest productivity drop: organizations with two development sites in our database were 25 percent less productive on average than those with just one. It is no coincidence that many of the world's best software development organizations chose to concentrate more than 90 percent of their software developers in a single location.

## Tailoring agile across the software stack

Many of the biggest benefits of the agile approach arise from close cooperation between developers and the end customer. As a result, the development of end-user applications has been the principal focus of many agile efforts, and organizations often find it easiest to implement agile methods in the development of their own application layers. Most software systems are built as a stack, however, with a hardware integration layer and one or several middleware layers below the application layer. Companies sometimes struggle to understand how they should best apply agile techniques, if at all, to the development of these lower-level layers.

In practice, the most successful organizations take a selective approach. They pick and adapt a specific subset of agile tools and techniques for each layer in their stack, and they alter their development approach to take account of the way agile is applied in the layers above and below.

Middleware development, for example, with its slower evolution of requirements and emphasis on standardization, can lend itself to an approach in which individual development cycles or sprints are longer than in application development. While doing this, companies should strive for a pace that is synchronized with the sprints in the higher layers. Middleware teams will also take extra steps to ensure their test cases reflect the impact of rapidly changing applications driven by faster-moving agile teams.

At the level of hardware adaptation, however, freezing requirements early remains a priority to allow sufficient time for hardware development. Here, an organization may still find it beneficial to pick specific agile tools—like continuous integration, test automation, and regular production of prototypes—to capture the benefits in productivity, time to market, and quality they provide.

## Rolling out agile development

For many organizations, the shift to agile software development represents a significant change in approach and culture. Like all large-scale organizational change, a successful transformation requires care in planning, execution, and ongoing support.

Most organizations begin their change journey by assessing their current practices and developing a blueprint for improvement. This blueprint will define all the extra capabilities, new management processes, and additional tools the organization will need. These may include extra training for developers, investment in test automation infrastructure, and a clear approach for the management of release cycles, for example. The adoption of agile methods will have implications that go far beyond the software development function. These must be taken into account during the development of the blueprint. Companies can engage the wider organization in a variety of ways, from conversations with leaders in other functions to crowdsourcing-style suggestion schemes.

The blueprint is validated and refined using a targeted pilot in one or a few teams. This pilot serves several functions. It helps the organization

## Agile development

The term "agile" describes a collection of rapid, iterative software development approaches. Agile involves a wide variety of tools and techniques, with certain common elements at its heart. Tightly integrated cross-functional teams that include end-customer representatives ensure the product reflects real user needs. Those teams do their work in rapid iterations to refine requirements and weed out errors, with continuous testing and integration of their code into one main branch. Techniques like colocation of project teams, daily meetings, pair programming, and collective ownership of code promote collaboration within the team.

identify and adapt the tools and techniques that best suit its needs. It also helps developers, product owners, and managers develop the skills they will need to apply and teach—in the wider rollout. Finally, the pilots serves as a demonstration to the rest of the organization of the potential power of the approach.

As an organization moves from its initial pilot phase toward a wider rollout, the need for a long-term perspective and strong top-management support is particularly critical. This is because most companies experience an initial drop in productivity as their developers, product owners, and managers get used to the new way of working. Analysis of our benchmarking data suggests that this drop is usually around 14 percent at its deepest before productivity recovers and goes on to surpass pre-agile levels by 27 percent or more.

Beyond preparing themselves for the inevitable initial dip in productivity, the best organizations take steps to preempt it. One key step is ensuring that the right engineering practices, capabilities, tools, and performance management mechanisms are in place as the rollout commences—the subject of "Teaching elephants to dance (part 2)."

Adopting the approach described here and in the following article has delivered remarkable software development performance improvements to some large companies. One North American development organization, for example, created a modified version of the agile "scrum" approach to improve the reliability of mission-critical software delivery. The company rolled out the approach to more than 3,000 developers using a large-scale change management program including training, coaching, communication, and role modeling. The results of this effort exceeded the company's original expectations, with throughput increasing by more than 20 percent, significant cycle time reductions, and higher customer satisfaction.

□ □ □

Applying agile in larger and more complex efforts requires modifications to standard agile methodologies. Specifically, organizations need to create structures and practices to facilitate coordination across large programs, define integration approaches, ensure appropriate verification and validation, manage interfaces with other enterprise processes (e.g., planning and budgeting), and engage with other functions (e.g., security and infrastructure).

*The authors wish to thank Florian Weig for his contributions to this article.*

*The article was first published by McKinsey's Operations Extranet.*


04

# Teaching elephants to dance (part 2): Empowering giants with agile development

Peter Andén, Santiago Comella-Dorda, André Rocha, Tobias Strålin

In big companies, lightweight development methodologies require heavyweight support behind the scenes for maximum benefits and minimal cost.

In "Teaching elephants to dance (part 1)," we showed how the agile development methodology produces better, more reliable software faster and at lower cost than traditional approaches. Capturing such benefits in large organizations, however, has a price: it requires companies to establish additional governance structures and accept a short-term loss of developer productivity. In this article, we look at the key practices that large companies must master to lessen the challenges of managing agile development and deliver the most value the approach can provide.

All product development processes have the same broad objectives: to deliver the right features to the customer at the right time and in the right quality, all at the lowest possible cost. For companies implementing agile at scale, doing that requires the right design choices, practices, and processes in four broad categories: products and architecture, methods, organization, and enablers.

## Products and architecture

This category is focused on creating the appropriate scope for each product—within the context of the right overall product portfolio—as well as building an architecture to support it.

Product requirements. Adherence to the agile principle of simplicity helps ensure that individual teams remain focused on objectives. Leading companies use the idea that each agile team has only one backlog and one product owner to avoid duplication, conflicts, or "scope creep." By doing this, they make certain that product requirements are clear and responsibility for the delivery of each requirement is ultimately allocated to a single team.

The product owner plays a critical role in this process. During early-stage testing, for example, the PO will collect and filter feedback from end users, decide which requests should be added to the backlog of feature requests, and allocate those requests to the appropriate teams. Typically, the backlog is structured along several different levels of granularity, starting from the original requirement as formulated by the end users and subsequently broken down into smaller and more detailed requirements, often called user stories. To enable coordination across multiple teams, PO organizations often follow a similar structure with a hierarchy of POs, senior POs, and chief POs owning backlogs at the team level, program release level, and portfolio or suite level.

Architecture. A robust modular architecture is essential to the success of large-scale agile projects. Modularity aids the division of work between teams, minimizes conflicts, and facilitates the continuous integration of new code into the product for testing and evaluation. Under the agile methodology, the rapid evolution of the product during development sprints makes it difficult to fully define such architecture up front. Instead, leading companies reserve development capacity specifically for modularization work. This means developers can concentrate on the delivery of features and refine the surrounding architecture as they go.

In addition to agile teams' work on the architecture of their own modules, a dedicated cross-project

architecture team can help manage the major interfaces and dependencies between modules.

To minimize dependencies and the resulting waiting times, agile teams are usually formed around the delivery of features (rather than around modules or components). It is likely then that individual teams will end up working on multiple modules, with more than one team potentially contributing to the development of individual modules. While this approach creates the possibility of code conflicts, which must be managed, it also promotes close coordination between teams and encourages the development of simpler interfaces between modules. Leading companies consider the trade-offs between simplicity and the potential for conflicts when allocating development resources, so some particularly complex or critical modules may get their own dedicated teams.

## Methods

Software's short life cycle means that testing and integrating changes is an ongoing process. This has implications for the approaches an organization takes to software development.

Test-driven development. The rapid, iterative nature of agile development makes maintaining quality a challenge. Companies who do this well are adopting test-driven development methods that help them accelerate the development process by increasing the chance that the software is right the first time. Under the test-driven development approach, agile teams begin by writing test cases for the specific features they are implementing. Then they write product code to pass those tests. Through the development process, the tests are updated alongside the code, and every iteration must pass the test prior to release.

Beyond accelerating the test process, test-driven development has a number of other advantages. It helps make requirements clearer and more specific, since they must be built into the test protocols. It enables real-time reporting of the progress of the whole project, since managers can check the number of tests passed at any one time. It encourages teams to write simpler and more rigorous code that focuses on meeting user requirements. Finally, the availability of the test protocols simplifies future updating and code maintenance activities.

Continuous integration. Early and regular testing requires access to the latest version of the product under development. To avoid labor-intensive and potentially error-prone manual recompiling and rebuilding, best-practice companies support their modular architecture with a continuous integration infrastructure that makes regular (daily or every few hours) builds of the product for testing and use the latest version of code released by the development teams.

Some very mature agile development organizations will make these daily builds of their product available directly to customers with the confidence that their test-driven development and continuous integration processes will ensure sufficient quality and reliability. Such a rapid release cycle is not always desirable, however. It is more common for the PO to make a release only when there is sufficient new functionality available in the product. In addition, some organizations have one or two "hardening" sprints before each scheduled release, in which teams focus on improving product quality and performance, rather than adding new features.

The systematic use of agile practices like continuous integration and test-driven development leads to quantifiable benefits in the quality of the software developed.

## The organization

The key to large companies' ability to be agile in software development lies in the way they structure their organizations, how they manage and support the teams within the organization, and how the organization interfaces with its partners.

Team coordination. The need to keep multiple teams coordinated is the most significant difference between agile in large and small projects. Getting this right requires mechanisms for strong coordination and monitoring of code conflicts. Strong coordination starts by holding regular meetings within each agile team (typically every day) and among the various teams in a project (usually two to three times per week). To monitor and minimize code conflicts, two important steps can be taken. First, the product architects can monitor system interfaces and the impact of changes. Second, dedicated owners can be assigned to each product module to continuously monitor and assess the quality of the code.

Protect the team. In order to effectively protect their development teams, the best companies manage the various essential interactions between the teams and the rest of the organization. This includes staffing teams appropriately from the beginning, so they have all the capabilities they need to complete their current sprints without additional resources. It also involves procedures to ensure teams complete all the testing and documentation required to comply with corporate standards and security requirements.

In large and complex projects, development teams can easily be distracted by requests from users, managers, and other teams. Taking steps to minimize such distractions during development sprints allows teams to focus on achieving the objectives of the sprint. At the top of this shielding infrastructure is usually the PO, who prioritizes requests for new features or product changes. The PO will be assisted by a dispatch team, which is responsible for the incoming stream of bug reports and minor change requests arriving from users, field test teams, and other stakeholders. The dispatch team will eliminate duplicate requests and validate, categorize, and prioritize issues before adding them to the backlog and allocating them to the appropriate team. In many project organizations, the project manager will collaborate with the PO and support shielding the teams.

Finally, companies establish clear interfaces with relevant parts of the wider organization, so development team members know where to go for advice on the company's graphic design standards, for example, or to check that products and features will meet legal requirements in all relevant markets. A useful construct is to appoint a single point of contact (SPOC) from each relevant organization. The SPOC is required to attend the release and sprint planning meetings and reviews to ensure appropriate coordination and engagement while limiting the additional load on central functions.

Managing distributed teams. Agile development works best when all developers sit at a single location. In many organizations, however, such colocation is not possible. Their development teams may be sited in different countries, for example, or some parts of the development may be outsourced to external organizations.

To make agile work well in distributed environments, companies must make further modifications to core agile practices. Keeping multiple teams coordinated, for example, may require additional up-front planning prior to the start of development sprints. The sequence of development activities requires extra care too. Focusing early on aspects that will have significant implications for many teams, like the architecture of the product or its user interface, helps ensure consistency later on.

The best companies also work hard to facilitate communication between distributed teams. They do this using virtual communication tools like videoconferencing and Web-based document management and sharing tools. They also facilitate visits, exchanges, and face-to-face meetings between teams where possible. Requiring more detailed documentation as part of each agile sprint also helps subsequent teams to understand and build on work done by distant colleagues. The dedicated effort required to document this can be minimized with the use of automated documentation generators.

Partnering with vendors. Even companies that run successful internal agile processes frequently fail to apply the same principles in their interactions with other vendors. Rather than investing time and effort negotiating traditional—and inflexible—contracts that aim to capture all the requirements of the project up front, some leading players are now working with external suppliers in the same way they do with their internal agile teams. By encouraging collaboration and a focus on output, this approach aligns internal and external development efforts and promotes greater efficiency for all parties involved.

## Enablers

With organizational structures, product scope and architecture, and methods in place, large organizations will need to equip themselves with the know-how required to expertly implement agile software development. Building capabilities, tools, and performance management systems will enable teams to perform at their best.

Capabilities. As they scale up their agile transformation, companies need to dedicate special attention to developing the right capabilities across their organization. Agile places new demands on software developers, who may have to learn to operate in a less specialized, more flexible, more self-reliant, and collaborative environment.

Leading companies promote multi-skilling in their development teams. Typically, individual developers will have one or two core areas of expertise but will also acquire skills in related areas. Multi-skilling helps the development teams adjust to inevitable workload changes and other skills required during the project. Combined with collective code ownership, it also helps different agile teams work independently. Multi-skilling also works well for developers, giving them plenty of opportunities to upgrade and extend their skills, and it facilitates communication and collaboration with colleagues working in different areas. Last but not least, having top-notch developers usually makes a big difference in team productivity.

Agile places new demands on managers too, particularly product and R&D line managers, whose role under agile may change radically. Product managers need to operate much closer to the development engineers, prioritizing and explaining the work that needs to be done. Similarly, the most effective line managers in agile software development environments will focus on enabling their engineers to do what they are best at: developing new products. The role of the line manager is to ensure that the development team holds the required capabilities, a high motivation level, and a strong “can-do” mindset. Importantly, line managers also make certain there are no impediments to development progress. It is vitally important to support managers through this transition, but it is frequently ignored.

Good capability-building efforts make use of a range of methods, with classroom learning supported by extensive on-the-job coaching, mentoring, and support to reinforce the use of new practices.

Tools. A common development tool chain across all agile teams is an important element of effective project execution and control. This needs to be in place from the beginning of the agile rollout. Examples include technical tools for automated testing, quality analysis, configuration management, continuous integration, fault reporting, and product backlog management systems.

With these tools, companies can mandate the adoption of certain agile methods right at the start of the transformation process. For example, they can ensure that testing takes place from the beginning of each development sprint to catch as many issues as possible before code is released into production. They also are able to ensure code actually is released into production at the end of each sprint to continue the rapid identification of issues.

Performance management. Good management isn't all about IT systems and tools, however. Leading companies also make extensive use of visual management techniques—another core agile practice—with teams using whiteboards that show the status of and pending actions associated with their current sprint. Not only do these boards serve as the basis for their daily meetings, they also allow product owners and members of other teams to see what is going on at a glance.

Finally, companies need to balance the independence and flexibility of agile teams with the need to ensure the wider organization has a clear understanding of their progress, and can intervene to provide extra support when problems occur. Best practice is to do this by adopting standard systems and processes for performance management while using clear and closely tracked metrics, including engineering productivity indicators.

□ □ □

The agile approach has proved greatly effective in improving the speed, productivity, and responsiveness of software development. Applying a methodology that was developed for small teams across a larger organization requires companies to make some specific changes and additions. Adopting tools and practices described here allows even the most complex development projects to capture the benefits of agile.

*The authors wish to thank Florian Weig for his contributions to this article.*

*The article was first published by McKinsey's Operations Extranet.*


05

# From box to cloud: An approach for software development executives

Santiago Comella-Dorda, Chandra Gnanasambandam, Bhavik Shah, Tobias Strålin

As the world moves to cloud-based software, many software development executives wrestle with transitioning from packaged to cloud products. Pointers from successful software vendors can ease both the decision and ultimately the move.

Recent growth in cloud-based software as a service (SaaS) is expected to continue at 20 percent each year through 2018, when the global market could reach nearly $85 billion. Switching from packaged or "on-premise" software to SaaS has a number of benefits that include improved user experience and lower delivery and support costs. It also enables companies to access new markets and incorporate innovative third-party cloud software.

At this point, however, SaaS remains something of an afterthought in the portfolios of leading software vendors. According to one report, only 8 percent of the revenues generated by the top 100 software vendors comes from SaaS models—and seven of the ten biggest companies draw less than 5 percent of their software revenues from SaaS. Other research shows that the SaaS penetration in most software app categories remains low today, ranging from 1 to 36 percent. By 2018, however, its share should increase materially, achieving up to 72 percent penetration with some apps. While many vendors have yet to jump onto the SaaS wagon, a few that have been delivering SaaS experiences for years are busy upgrading their technical architectures to implement the latest generation of cloud technologies. These include new persistence and database models (in-memory or NoSQL databases, for example), faster analytical platforms, adaptive user interfaces, and elastic computing, among many others.

As companies attempt to transition packaged software to SaaS or upgrade existing SaaS solutions to leverage new cloud architectures, they often face a number of challenges. Conversations with senior software development executives surfaced a number of concerns and questions regarding this transition.

What kind of cloud architecture should they target? Should developers use public infrastructure-as-a-service (IaaS) or platform-as-a-service (PaaS) solutions or choose the private cloud? Do they need to rewrite their entire code base?

How does the organization manage the transition to its target state—from legacy architecture to cloud-based services-oriented architecture? How long should the transition take and what are the key steps? Should the company wait until cloud and on-premise products achieve parity?

What changes should be made to development and operating models? Should development methods be changed? How could this shift affect software release cycles? Will the company have to change the way it engages with customers?

What capability and cultural shifts does the organization need? How should a company build the necessary talent and capabilities, what mindset and behavioral changes do they need, and how do they select the right development, IT, and infrastructure approaches?

A deliberative process begins with a careful consideration of which code base type, architecture modification, and cloud infrastructure is most appropriate. Then—to ensure successful execution on the choice made—software executives will need to make several commitments

regarding the scope of the product, the approach to development, and the allocation of resources.

## Choosing the right approach

Software companies who are considering making the switch to cloud software face three critical decisions, and their optimal way forward will depend on their main objectives and starting points.

The first decision concerns whether it makes more sense over time to choose a unified code base for both packaged and cloud software or to have a separate one for each. Ultimately, this decision comes down to a few key factors. First, the organization's long-term vision is important when determining the ultimate purpose of the application. Is the team trying to build an optimized application for the cloud or is it attempting to leverage specific benefits of the cloud while providing additional options to customers? The second issue concerns the maintenance costs for two code bases. In this case, how long does the company plan to continue with both packaged and cloud software products, and is feature parity required? For many software vendors, it seems likely that packaged software suites won't go away anytime soon. The final factor involves talent and culture. Does the team have the desire and attitude required to learn new technologies and unlearn past coding practices?

When a unified code base makes sense. A unified code base might be preferable if current customers view the cloud as just another channel. That is, the company does not expect all of its customers to transition away from on-premise software in the short to medium term (see text box "When less is enough"). Or the company might not need best-of-breed cloud architecture to take advantage of the basic cloud benefits (including elasticity, scalability, and low cost). A unified code base works when the company has to maintain and manage multiple versions of the product. From a practical standpoint, another reason to choose a unified solution is that a company has evidence that its development teams are willing not only to learn new technology but to unlearn past coding practices as well.

When to choose separate code bases. Maintaining separate code bases for packaged and cloud software may be ideal when managers see the cloud as the key channel for future growth and expect to phase out the on-premise product. If customers expect the cloud-based product to be different in terms of look and feel compared with the desktop version and also expect it to include features provided by other cloud-based offerings (weekly releases, better scalability, and support for social tools, for instance), then separate code bases may also be the right choice. Other reasons to opt for separate code bases may be the fact that the company doesn't have to manage feature parity between both cloud-based and on-premise products, since it will soon phase out the latter or the software team must completely rethink the user experience and has the required skills to execute.

The second critical decision: companies can choose to refactor and "re-architect" on the go or build an entirely new architecture. When making this decision, leaders should consider two factors: the viability of the current architecture given the projected road map of the company's software products and the time-to-market requirement.

When to refactor. Refactoring is typically much faster and preferable if the current architecture might not be ideal for the cloud architecture but does have basic structural elements such as identifiable layers. It also makes sense if developers can port multi-tier applications to cloud architecture without undertaking a complete rewrite. Another scenario for which refactoring may be the better choice is when the company needs to release the first cloud-based version as soon as possible. Fully refactored, services-based architecture can help drive frequent and small releases but is not a necessity to get started with the cloud-based product.

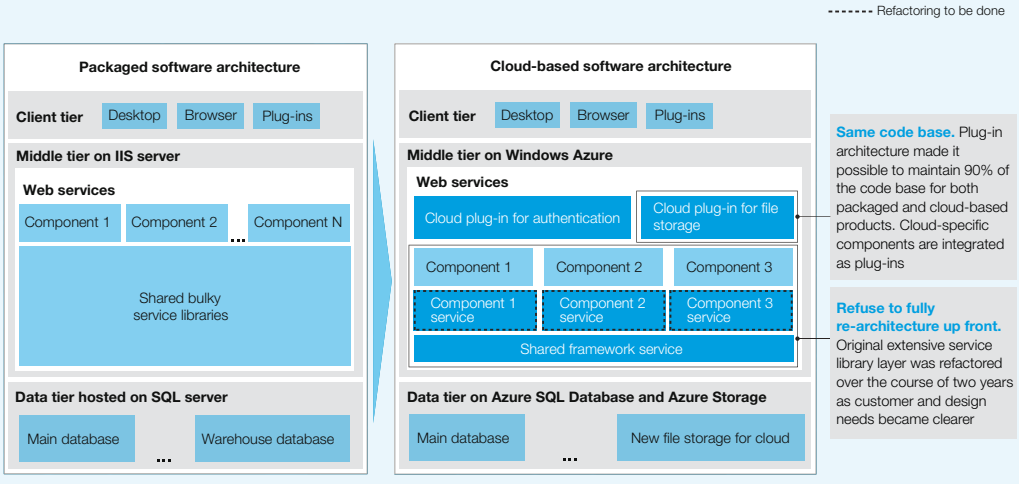
When to design a new architecture. Developing a new architecture makes sense if the current

## When less is enough: Software's measured journey to the cloud

One software company released the first version of its application life cycle software suite nearly a decade ago. As is typical of many packaged applications, the product had a two- to three-year release cycle. Five years later, this company began to develop a cloud version to achieve some of the benefits the technology provides, such as scalability, elasticity, ease of deployment, and minimal up-front investment for customers. As the software team began its migration journey from packaged to cloud software, it made two key decisions:

**Use a single code base.** The packaged version of its software will have a significant customer base going forward. The team decided early on that it would use the same code base for both products and adopt a plug-in-based architecture for cloud-specific components. This decision allowed them to utilize 90 percent of the code base for both versions of the software.

**Refactor as you go.** The packaged version is a three-tier application with the server running on the Windows platform. The product has a services-based architecture, but the services were not modular enough for a good cloud-based application. Since the team had not created the product in the cloud, it had difficulties making the transition. Team members chose refactoring in order to build a "minimum viable product" for the cloud and then continued refactoring existing code base after releasing the product.




Source: McKinsey analysis

Team leaders highlighted several lessons they learned along the way. For instance, the use of advanced engineering systems and the team's "can-do" attitude were big transition enablers. They also learned that cloud-based products require three to four times more diagnostics capability compared with packaged software. Finally, they noted that the customer engagement model can be very different when product releases take place every three weeks instead of every two years. Today, approximately two million developers use this software company's cloud version.

design is just not suitable for the cloud. For example, it might be a monolithic architecture that demonstrates symptoms of "spaghetti" code. Another consideration is the architecture's scalability. Sometimes an architecture originally built for on-premise isn't really designed to scale up to a larger number of users or it does not support multi-tenancy. Companies often build up related "technical debt" because of prior architecture decisions. For example, a payroll processing company decided to overhaul their current architecture to be able to move to open stack since portability is a key requirement for them. The company built some of the new system from the ground up while tactically leveraging stable calculations engines and other components. Even while re-architecting much of the stack, the company didn't update a few mature components, including some on mainframe, since the risks of updating those components outweighed the potential benefits.

Another critical decision is the choice of public versus private cloud infrastructure. Companies can build their products on top of either privately hosted platforms or public IaaS or PaaS ones that rely on a service provider. This decision primarily concerns economies of scale, since the scale of infrastructure deployment, the company's tolerance for risk (data security or performance issues, for example), and regulatory requirements will ultimately drive it. IaaS platforms provide flexibility and control but entail the trade-off of additional complexity and the up-front effort required to build a user-ready service for them. Conversely, PaaS platforms often offer many capabilities that can help companies accelerate the transition to the cloud, but these platforms generally include proprietary or vendor-specific capabilities. As such, they require software created for a specific vendor's platform and stack, thus locking in those suppliers. While a small degree of vendor lock-in does exist with IaaS systems, it is relatively easy to plan around those areas.

When to go for a private cloud. Private clouds work when the developer has sufficient internal scale to achieve a comparable total cost of ownership to public choices. That typically means it employs tens of thousands of virtual machines (VMs). It is also the right choice if at least one of the following four considerations is critical for the specific system or application and therefore precludes the use of the public cloud: data security, performance issues, control in the event of an outage, or technology lock-in. A final factor involves regulatory requirements that might restrict efforts to store customer data outside of set geographic boundaries or prevent the storage of data in multi-tenant environments.

When to choose the public cloud. Developers should consider the public cloud approach if the project lacks sufficient scale (will not involve tens of thousands of VMs, for example) or a high degree of uncertainty exists regarding likely demand. Using a public cloud is a more capital-efficient approach, since building a private cloud requires significant resources that the company could probably invest more effectively in the mainstream business. Another reason to go public: the system or application is latency tolerant. Experience shows that the latency levels on public clouds can vary by as much as 20 times depending on the time of the day. It also makes sense—if there are no specific regulatory requirements—that applications store the data in a particular place beyond performance needs. Even if companies decide to use a private cloud for their most critical applications, many decide to use public cloud for certain more basic use cases (dev/test workloads, additional temporary capacity, for instance).

## Six cloud-hopping design principles

Once executives have made their code base, architecture, and infrastructure decisions, they begin developing their cloud-based software. To better understand how software players successfully make the transition, McKinsey reviewed a number of external cases and

conducted in-depth interviews with leading software players. Those who succeed in the journey from on-premise to cloud software development share six commitments.

Shoot for the minimum viable product instead of feature parity. Organizations moving products to the cloud often discover that achieving full-feature parity could take several years. Instead, successful vendors often decide to release a minimum viable product (MVP) to customers in six to nine months. This strategy allows them to test their architecture and functionality quickly. The approach also forces them to think deeply about which types of product functionality deliver sought-after core customer experiences and what they have to emphasize to get that functionality right. By putting a workable MVP with the most important features in user's hands as quickly as possible, the team is able to both gather crucial initial customer feedback and rapidly improve on their cloud-based development skills.

Treat users as part of the day-to-day development team. Developers need to engage with their customers early and often—and shifting to a cloud model opens new ways of interacting with them. Teams can get feedback from customers in near real-time as soon as—or even before—they release a feature. User engagement also allows developers and product managers to ask customers to prioritize their needs via blogs when the product is in the concept phase and provide a basic product to specific early adopters. They can then codesign the full-featured version with them. Experience shows that collecting early feedback can help teams shape how they prioritize the features that are still in development.

Running the application centrally for all customers also opens up new capabilities. Developers can, for instance, employ logging and analytics to understand customer actions, taking a highly data-driven approach to tracking their usage patterns. Likewise, performing "A/B" features and functionality testing gives teams a data-driven approach to decision making. The 2012 Obama presidential campaign in the United States, for instance, used about 500 A/B tests on its Web page interaction, copy, and images. The approach increased donations by nearly 50 percent and sign-ups by over 160 percent.

The cloud also enables teams to roll out functionality in a controlled manner (first 1 percent of customers, then 5 percent if all goes well, then 10 percent, etc.).

Expect and tolerate failures. Cloud infrastructure brings many benefits including the ability to grow or shrink resources for an application in real time. However, the shared nature of cloud infrastructure can pose challenges because of factors beyond the developer's control, such as hardware or network failures or slowdowns. And, as with all customer data centralized in the cloud environment, developers need to design an architecture for the application that can accommodate these failures and work around them.

For success, companies will need to develop a mindset that accepts failures. Without it, developers will hesitate to make changes, making release cycles grind to a halt. One Internet content provider learned this lesson the hard way after experiencing service disruptions due to a third-party Web services provider failure. In response, the company made its applications more robust in the face of such disruptions. Now, if similar problems occur, their apps are designed to provide a somewhat diminished customer experience rather than a complete crash. On top of this, to simulate random failures, the company created a special tool in the form of a script that will indiscriminately kill infrastructure services. This approach enables it to test application responses against failures that may eventually happen. It also helps teams learn about challenges specific to cloud-based development and incorporate customer inputs early.

Adopt agile and DevOps approaches. Companies should adopt agile thinking and DevOps, a software development approach that focuses on product delivery, quality assurance (QA), feature development, and maintenance releases. DevOps builds on many "agile" concepts, like working in cross-functional teams and in short iterations all the way to deployment. Software executives also need to integrate their QA, operations, and security organizations with their R&D teams and schedule at least one release per month. Continuous integration, including integration into the main software branch, should be implemented with at least daily frequency. Releases should be scheduled as frequently as possible to ensure early user feedback. The release cycle can range from several releases per day to one per month. Once code base is refactored into granular services, it is possible to achieve very short release cycles without destabilizing the entire product base.

The need for this shift goes to the heart of the differences between packaged and cloud software. With packaged software, releases are expensive because teams have only one true chance to launch a product. Consequently, releases occur once or a few times every 24 months. In a cloud environment, in contrast, most vendors find that incremental releases reduce the complexity of deployment and the magnitude of potential failures at the time of release. The incremental release approach leads to dozens of small releases for an individual product in its lifetime.

Give developers QA and testing responsibility. Another hallmark of successfully moving from packaged to cloud software is the choice of companies to hold their software developers—not the code testers—accountable for quality. These companies seem to blur the boundaries between development and QA roles. The idea is to allow the software developers to resolve critical issues immediately as they become apparent. This approach requires them to deploy critical fixes continuously in addition to running short release cycles. It makes sense: despite the iterative nature of agile software development, cloud users will not accept or use apps with significant unaddressed issues. And it's also efficient: a developer can fix a bug introduced just two weeks ago much quicker than one that was introduced six months or two years back.

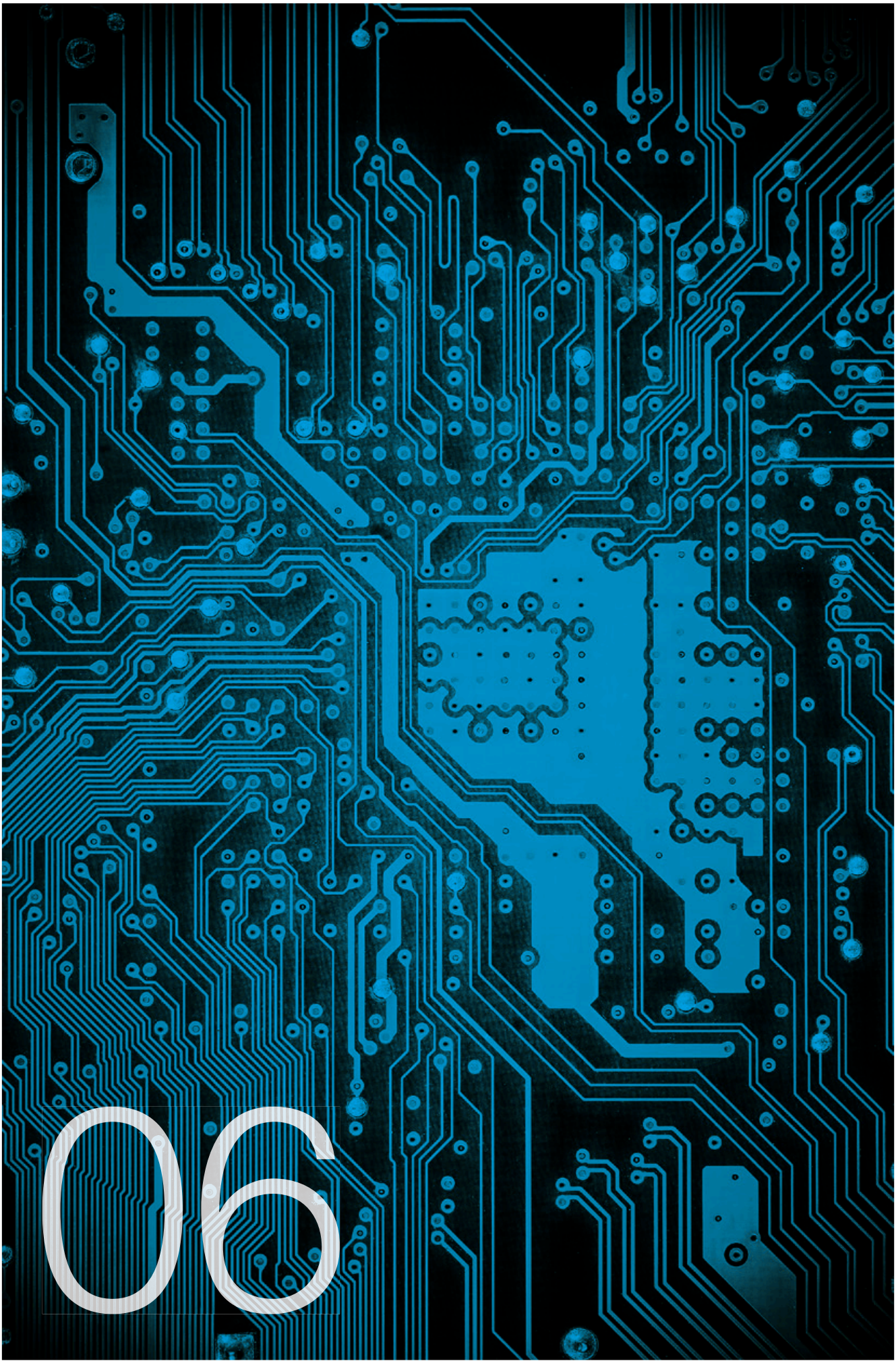
Another very important factor is that developers understand that fixing SaaS issues is fundamentally different from fixing problems with packaged software, which requires them to adopt new practices. It is normal, for example, to expect customers to take their servers offline to debug a problem for on-premise software. This is not an option for service-based software, since customers across multiple time zones are using the service. Building advanced diagnostics and tracing capabilities in the software is much more imperative for cloud-based software. Another similar example is NoSQL database adoption, which requires a significant unlearning of how developers worked with traditional, relational databases.

Invest in cutting-edge capabilities and automated test environments. McKinsey's observations of successful software developers suggest that hiring top development talent who can inject new external expertise into the organization at the operational and management levels is critical to making the switch to cloud software. Another crucial enabler is investing in tools and infrastructure to power the cloud-focused development model. The organization should shift all build, integration, and testing operations to a continuous and automated model that supports rapid release cycles. Leading cloud software providers—such as some of the world's largest search and social media companies—regularly build and test the entire code base several times a day. Companies can reduce these intervals to as little as 15 minutes, but doing so requires a very advanced IT environment. Leading cloud players provide an environment where developers can test code changes against any of the portfolio products that could be potentially affected. This enables designers to conduct solid integration testing on their own before submitting the code for real integration.

Even if it generates significant buzz, cloud-based SaaS remains a relatively small part of most leading software developers' product portfolios. As the share of cloud-based software grows, developers will need to increasingly focus on transitioning away from packaged, on-premise software. Reaching carefully considered technology decisions and committing to several organizational and operational approaches to developing software as a service can help developers successfully transition from packaged to cloud software.

*The authors wish to thank Buck Hodges, Engineering Director for Microsoft Visual Studio Cloud Services, and Roberto Masiero, VP ADP Innovation Labs, for their contributions to the article. The authors would also like to thank Akash Shah for his contributions.*

*The article was first published by McKinsey's Telecom, Media, and High Tech Extranet.*


06

# Complexity costs: Next-generation software modularity

Peter Andén, Simone Ferraresi, Tobias Strålin

As software becomes more sophisticated, it gets increasingly expensive to customize, maintain, and extend. A new modularity approach can turn the tide of rising costs and risks, allowing companies to unleash the full potential of software in their businesses.

Software has become a key differentiator for the advanced and high-tech sectors, and it now plays a central role in many aspects of business. Companies rely on sophisticated software applications behind the scenes to design their products and execute their processes. Products, from communications satellites to cars and washing machines, use embedded software to deliver the features customers want, and which would be hard to implement in any other way.

But as its scope and the range of its application are continually stretched, software is getting increasingly complex. This creates new headaches for manufacturers. First, complex software takes ever more time and resources to develop. Second, companies face the challenge of supporting multiple variants of their software in use. Finally, multiple layers of customer additions and modifications create the need for lengthy tests to prevent unforeseen and undesirable effects on other parts of the system, making it harder for companies to promptly respond to new user requirements, to regulatory changes, or to the opportunities presented by new technologies.

## The first line of defense

For many companies that make application and embedded software, modularity represents the first line of defense against software complexity. Modularity, both in hardware and software, is a widely accepted engineering principle that promises three major benefits. First, it reduces the up-front effort of product customization to suit different customer needs, limiting the required changes to a small number of modules. Second, it allows a richer product portfolio while limiting the costs and drawbacks of added complexity. Third, it makes products faster and easier to maintain, as any change required to fix problems or introduce new functionality should be limited to a small part of the code.

## Why modularity strategies still fail

Many software producers still struggle to consistently capture and sustain the full benefits of their modularity strategy. This happens for a few common reasons. Sometimes, the original design is flawed. Drafting a modular architecture without fully considering future customer requirements can mean companies have to implement simultaneous changes to multiple modules in order to incorporate customizations or new features. Or they may need to repeatedly change and recertify the same few modules to meet different customer requirements.

In other cases, a good design is weakened over time, as additions and changes to the software are made without ensuring their compatibility with the existing modular architecture. Such “quick fixes” may appear cheaper in the short term, but they quickly damage flexibility and erode the self-contained nature of modules, driving up costs over the long run.

In still another common scenario, a company may use a good modular architecture badly, repeatedly creating entirely new versions of the software for particular purposes or to meet specific customer needs. The need to keep

all those variants up to date results in high maintenance certification and upgrade costs.

Any of these mistakes can have a major impact on manufacturers' ability to deliver products profitably and within deadlines, and to the cost of maintaining them in service. Compounding these issues, companies may be nervous about making the tough short-term changes that will deliver a more robust and easier-to-support product in the long run, preferring to stick with the "devil they know" rather than embarking on the expensive and potentially risky process of a major redesign.

## Making modularity work

Some companies are changing their software development methods to promote modularity strategies that work the way they are supposed to and that go on working through the life of the software. While their precise choice of software development practices and tools will vary depending on the size and culture of the organization and the nature of the software, McKinsey has observed that the best organizations maintain a focus on three important levers: a robust scalable architecture; a phased implementation approach; and a well-governed development organization to roll out and enforce this "new generation" modularization strategy.

### Robust, scalable architecture

Effective modularization means more than just cutting a piece of software into chunks. Modularization must simultaneously balance multiple, sometimes conflicting, objectives: for example, weighing the simplicity of smaller individual modules against the need to manage more interfaces between them. New generation modularization strategies adhere to a number of core principles.

Functional decoupling. Removing unnecessary links and dependencies between functions is a critical enabler for many of the benefits of modularization. To do this, leading companies break down the use case that defines their software into a number of elementary functions. They then design their modular architecture so that each of these functions can be modified with little—or ideally no—impact on other functions.

Rationalization. Strong architectures control product proliferation before it starts, by agreeing which parts of the software will be standardized for all customers and where and how customization will be allowed.

Adoption of standards. Standard solutions, from data exchange to development methodologies, help simplify many aspects of software design and support. However, standards only work if they are applied consistently across the product, and if the product uses the same standards everywhere. The best players, therefore, make considered choices about which standards to adopt as a core part of their software architecture. Then they stick to them, limiting the ad hoc adoption or proliferation of inconsistent standards that might otherwise occur.

Technology rationalization. As with standards, strong product architectures require strategic decisions about which technologies (for example, programming languages, databases, and development tools) will be adopted. Limiting the number of technologies they adopt helps companies control the breadth of skills their development teams need in order to support their products and makes it easier to move developers between different parts of the product according to demand.

Scalability. The need to serve customers of widely different sizes is one cause of product proliferation. By building scalability into their product architecture (for example, ensuring that their standard software can run on simpler low-cost computing platforms, perhaps with a smaller set of features) companies can reduce or eliminate the need to produce different products just to meet different scale requirements.

Upgrade flexibility. Software upgrades can be time consuming and disruptive if the system has

## Gaps and overlaps: Efficiency gains from making modularity work

Modularity alone doesn't lead to product development efficiency. Understanding the interplay between standardization and custom design is key. The companies described in the examples below needed to shine a light on modification patterns and frequency to get the most out of modularization.

**Modular imbalance.** A major player in control software discovered that a single user interface module in its current product was involved in more than 90 percent of the product's requirements, making it almost certain that the interface module would need to be altered whenever user requirements changed. The issue was solved by breaking this interface module into a number of smaller modules—some of which managed the data presentation to the user while other "service" modules handled data analysis and communications with the underlying system. After the change, no single module accounted for more than 10 percent of the requirements. Moreover, by making different services independent of one another, the new approach made it easier for the software manufacturer to sell different versions of the product, from basic to full-featured, by progressively adding different modules on a robust architecture designed to integrate such add-ons.

**Over-customization.** When a large systems manufacturer analyzed the structure of its core product, it found that more than 80 percent of modules required modification in every project, even though requirements changed in just a few processes. A full redesign of the product architecture resulted in a complete turnaround of the module taxonomy. More than 60 percent of modules became identical in every project, and only 5 percent required custom code. For the rest it adopted a combination of parametric and swappable designs to meet varying customer requirements with the minimum software development and testing effort.

to be shut down and replaced entirely at every upgrade cycle. The ability to upgrade software by changing only the affected parts of the code makes the upgrade process faster and more reliable, and it dramatically reduces recertification costs. Strong product architectures facilitate this by implementing the most commonly upgraded elements in separate modules.

Alongside decisions about the platforms, standards, technologies, and upgrade strategies, the other key foundation of a robust modular design is a thorough and detailed understanding of requirements. Mapping the individual requirements of their customers against the software modules that will fulfill those requirements can be eye-opening for companies looking to improve the modularity of existing products (see text box "Gaps and overlaps").

Each module defined in a product architecture also needs its own strictly enforced modularization strategy. There are four basic strategies manufacturers can adopt to determine the degree to which they will permit customization of individual modules and the way that customization will be executed:

- Identical modules are the same in all versions of the product and are never customized.
- Swappable modules exist in a number of predesigned and pretested "flavors." Products can be customized by selecting and including one or several of these modules.
- Parametric modules allow customization to be implemented by changing the value of predefined software switches or variables, either during the initial configuration or by the

user during operation (for instance, to adjust the software to local regulatory parameters).

- Custom modules always require customer-specific code to, for example, include the customer's branding or graphical standards.

As the cost of developing, supporting, and upgrading a module steeply rises the farther one goes down the list, a company should aim to allocate the simplest strategy they can to each module in the architecture.

### Phased implementation

Once a company has designed the desired architecture based on its modularization strategy, the next step is to implement it in its products. For completely new products, full modularization doesn't add significant extra cost of complexity at development time. In many cases, however, the product already exists in the market, and modularization strategy rollout must take on a different form. Planned changes to the software must take into account the existing development plan for the product and the need to update the existing installed base, as well as dependencies between modules. It is also important to consider payback for the up-front costs and effort required. By identifying and prioritizing the parts of the product that stand to benefit most from modularization—often those that currently require a lot of custom coding and development resource to meet user change requests—companies can make modularization pay off quickly. And they can use the resources they free up by doing so to accelerate later phases of the plan.

### Well-governed development organization

A successful modularization strategy extends beyond the architecture of the software code. The best manufacturers make changes to their development organization and management systems to support and enforce software modularization. During the implementation process, the organization should track the progress of its modularization plans, looking at the number of modules that have reached their target state and the rate of deployment of modularized software in the field, for example. Tracking the savings achieved through modularization is also important to ensure continued support from the wider organization and the availability of sufficient development resources to complete the project. Such tracking must take an organization-wide approach to balance the savings made in maintenance, configuration, support, upgrades and future developments against the up-front costs of changing the product architecture.

Over the long term, the development organization must also ensure that the modular architecture is maintained. The best organizations hold specific individuals in their development organizations accountable for this as "module owners," responsible for minimizing code conflicts in particular modules and ensuring that modules can deliver what is required by the rest of the organization while sticking to agreed standards. At a higher level, a "project architecture owner" will manage interactions between modules. Periodic architecture reviews help ensure that the architecture evolves in a systematic way in response to emerging technologies or new customer requirements.

## Impact

For companies that successfully enforce new generation modularity, the rewards can be significant and wide-ranging. When one global provider of safety-critical software systems embarked on such a project, it had multiple objectives in mind, including improvements to product flexibility, easier adoption of new technologies, and lower development, deployment, and maintenance costs. At the end of a three-month effort on its main product, the provider moved from a situation in which 85 percent of the modules contained custom code to a design that allowed the vast majority of customization to be provided by swappable or parametric modules, with only 5 percent of them

requiring full customization. Once this change was implemented, the organization was able to reduce product configuration costs by 10 to 15 percent or use this improved development efficiency to cut time to market by an even greater amount. In addition, the reduced complexity of its products led to savings of 15 to 20 percent in support and code maintenance. The largest rewards, however, came as the organization embarked on the development of new software capabilities and upgrades, where the stronger underlying product architecture helped cut costs by 40 to 50 percent. Overall, the modularization program cut software development and support costs at the company by a quarter.

Software complexity is valuable to customers, but it comes at a price to producers. Modularization allows developers to minimize the cost of customization by creating multiple, standardized product elements. Companies that adhere to a set of modularization principles, implement them mindfully, and create an organization that accommodates them can significantly cut their software configuration and maintenance costs.


07

# Organized for success: Restructuring hardware companies

Gang Liang, Christopher Thomas, Bill Wiseman

In-house software development is gaining the attention of hardware companies. Adopting one of four basic organizational structures can help them reap the benefits.

Software isn't just for purchasing or outsourcing anymore. Increasingly, companies whose primary focus has been hardware are exploring in-house software development as a way to reduce costs, improve time to market, and differentiate themselves from competitors. Many of these organizations—such as those in consumer electronics, medical devices, automobiles, appliances, and heavy equipment—are beginning to hire more software engineers to support the development of the integrated circuits that are at the heart of many of their products. Chipmakers are a prime example of this hardware-to-software shift. In the late 1990s, it was common for chipmakers to invest in one software engineer for every ten hardware engineers; today the ratio is closer to 1:1.5 or, in some cases, 1:1.

Across the board, hardware companies are pursuing software development primarily to meet customers' growing demands for more sophistication in and more support for the components they buy. Earmarking significant portions of their R&D budgets for software development is becoming routine, and some are already providing end-to-end software-based products for customers. But, while some market leaders have been at this for a while, other players are only just starting to take a closer look at how they build, use, and manage software (see text box "What are they building?" page 49). They face a number of challenges: software resources at hardware-oriented companies tend to be limited, and engineering talent can be scarce and hard to acquire and retain. Additionally, efforts to divert scarce resources away from a company's traditional core business toward software development may meet with internal resistance.

Hardware companies that choose to pursue a rigorous software development program will need to have the right organizational structure—one that enables them to motivate talent, control the R&D budget, launch products more quickly, and meet customer expectations. Some hardware companies have established a central software organization to support all business units, while others are struggling to keep up with "rogue" development efforts happening within various business units—each with its own software team. McKinsey's work points to four potential organizational structures that software-minded, hardware-based companies may want to consider to get the most from their existing development efforts and to make it easier to pursue new software R&D initiatives: completely decentralized, completely centralized, hybrid, and leveraged. Advantages and potential pitfalls are associated with each. The appropriate structure will differ for every company depending on existing talent, resources, and overall business objectives.

## Four ways to organize

We have seen 1,000-person companies make the transformation from one organizational structure to another within 12 months, but an effort lasting years is much more typical, particularly if the company is starting from scratch or having to make hard decisions about which groups to merge. In either scenario, a change in metrics and mindset will be required. Executives will need to develop mechanisms for tracking the productivity gains from their software R&D, and they will need to foster engagement and commitment to software development across the company.

Completely decentralized. Hardware companies with a number of different business units that have little or no business or technical crossover likely would find it easiest to pursue a completely decentralized software organization. In this structure, each of the business units funds and manages its own software group, and the unit's general manager retains the autonomy to deploy software resources where needed. In the 1990s, Intel's architecture business unit boasted a dedicated software organization that created homegrown development tools to support its x86 systems. Even today, the software group works closely with a number of third-party software vendors—Oracle and SAP among them—to optimize those applications for every generation of its central processing units. Intel also had separate software groups dedicated to its NOR Flash and i960 businesses. The NOR Flash software team built up a strong capability in device drivers, and the i960 software team focused on enabling Intel silicon to work well with third-party software and applications. There was almost no overlap in customer bases or operations among those business units.

The completely decentralized model works well as long as the businesses units and technologies remain independent. If, for instance, units are combined or new businesses emerge and need the same fundamental software and technologies being developed and managed in other groups, it makes less sense (operationally and financially) to duplicate efforts. One large OEM, for instance, created separate business units for two of its consumer products designed for two different market segments. There were separate software development groups for each unit, but the company eventually realized that development teams in both used largely the same package with only a few feature differences. The implication: the company was wasting its resources and needlessly creating conflict and competition between two groups of engineers.

Completely centralized. For hardware companies whose business units rely on all the same technologies, a completely centralized software organization will be most efficient and effective. Under this organizational structure, software development and technological expertise radiates from a central group—one that reports to the C-suite—removing potential redundancies and significantly reducing resource and development costs. Having one centralized software group allows the company to better manage all of its licensees and reduce development costs.

A completely centralized model also confers other benefits upon hardware companies, including a consistent approach to R&D planning, a standard set of software development and management tools, a common software development process and methodology, and comprehensive rules and standards for assuring quality and appropriately managing source code. By establishing this level of consistency across all business units, hardware companies can reduce their R&D costs and accelerate growth in new and critical businesses that may not otherwise have the funding or technical capabilities to pursue software development as a complement to their existing work. This centralized structure also may facilitate offshore expansion or development outsourcing by making it easier for them to manage global engineering resources or maintain relationships with vendors.

There are a few drawbacks, however. For instance, the funding model for this approach can be complicated. In many companies that use a completely centralized model, the business units pay a "tax" based on their needs, financial strength, and other criteria. This can be a headache for the finance team, which has to calculate the difference between projected needs and actual demand for each business unit—each of which would obviously want to pay as little of this tax as possible. Additionally, under the centralized model, the business units would have less control over software development as a resource. Often, the objectives of the centralized software group and the business unit will not be completely aligned; the units may have unique requirements that a centralized organization simply may not be aware of.

## What are they building? The semiconductor software evolution

At the start of the shift toward in-house software development, many semiconductor companies were focused primarily on developing their own firmware—software embedded in their integrated circuits that would dictate how the chips would function. Over the past few years, some have started working directly with operating system vendors to make sure their device drivers will work seamlessly and their processors will perform optimally in those environments. Others began to release software tools (compilers, debuggers, tuners, and the like), plus common libraries and middleware, so third parties could create optimized applications for their company's chips. Most recently, semiconductor companies have started to create end-to-end, embedded software products for original equipment and original device manufacturers.

It is critical for companies that adopt a centralized model to pay attention to process, metrics, and collaboration—for instance, convening a small team that represents the interests of each of the business units and the centralized software group. The team would meet regularly to analyze software priorities and rank them according to business unit needs and the impact of certain projects on the company overall. It is also good practice to establish service level agreements between the centralized software group and business units to help clarify roles and responsibilities—and to preserve some level of control for the general managers.

Hybrid structure. This organizational approach combines the financial and technological efficiencies provided by a centralized software group with the greater flexibility and controls that a decentralized structure may offer the business units. At first glance, it seems to present the best of both worlds. In reality, there are significant funding and operational challenges to address. Under the hybrid model, the technologies and software capabilities that are common to all business units become the property of a centralized group, while the technologies and software that are unique to a particular business unit are maintained and developed separately.

Beyond just holding on to the common software, the centralized group should also establish best practices for its use and encourage sharing among all the other software teams within the company. To that end, a joint committee should be convened to manage common software development priorities, and service level agreements should be drawn up. But as with the completely centralized model, a charge-back process must be established; the use of common technologies would be subject to a tax based on revenues, profits, or other criteria, and each business unit's software organization would be required to fund its unique development initiatives separately.

Leveraged structure. Many hardware companies have a core business and a number of units that are derivative of the core. For instance, a company's core business may be manufacturing components primarily for the automotive market, but increasingly its technologies are also being used in medical and consumer applications. For such companies that are exploring market expansion, a leveraged software organization may make the most sense. Under this structure, the software group would report to the core business unit rather than to a centralized corporate team. As with the hybrid model, the software organization would own the completed software components and resources but would deliver them to the rest of the company. For instance, the software team in the company's consumer products business unit could take technologies developed by the software

team in the company's automotive unit and modify them to suit the business unit's and the market's needs. As with the centralized model, the core business's software group would need to establish best practices in software development and encourage sharing across the organization, but the other business units would have to fund their own unique development initiatives.

## Which model?

To determine which of these structures is best, companies need to consider their existing software capabilities—that is, the type of software R&D they are currently undertaking (if any), their overarching objectives relating to software, and the funding and other resources at their disposal. They should also consider competitor's software capabilities. Companies that already have, for example, lots of software R&D experience, or that have a core business unit with several businesses feeding off of it, will want to explore hybrid or leveraged models. The individual business units would immediately benefit from software technologies that are already in hand (managed by the centralized software organization), but they would retain the flexibility to create unit-specific products based on their unique technical and business needs. Such companies could see less duplication of effort and waste.

By contrast, companies that have limited software R&D experience may want to set up a centralized software organization focused on just one business or a few business units at first— starting narrow to ensure that success is within reach, but establishing best practices that can be rolled out more broadly as software development initiatives gain momentum.

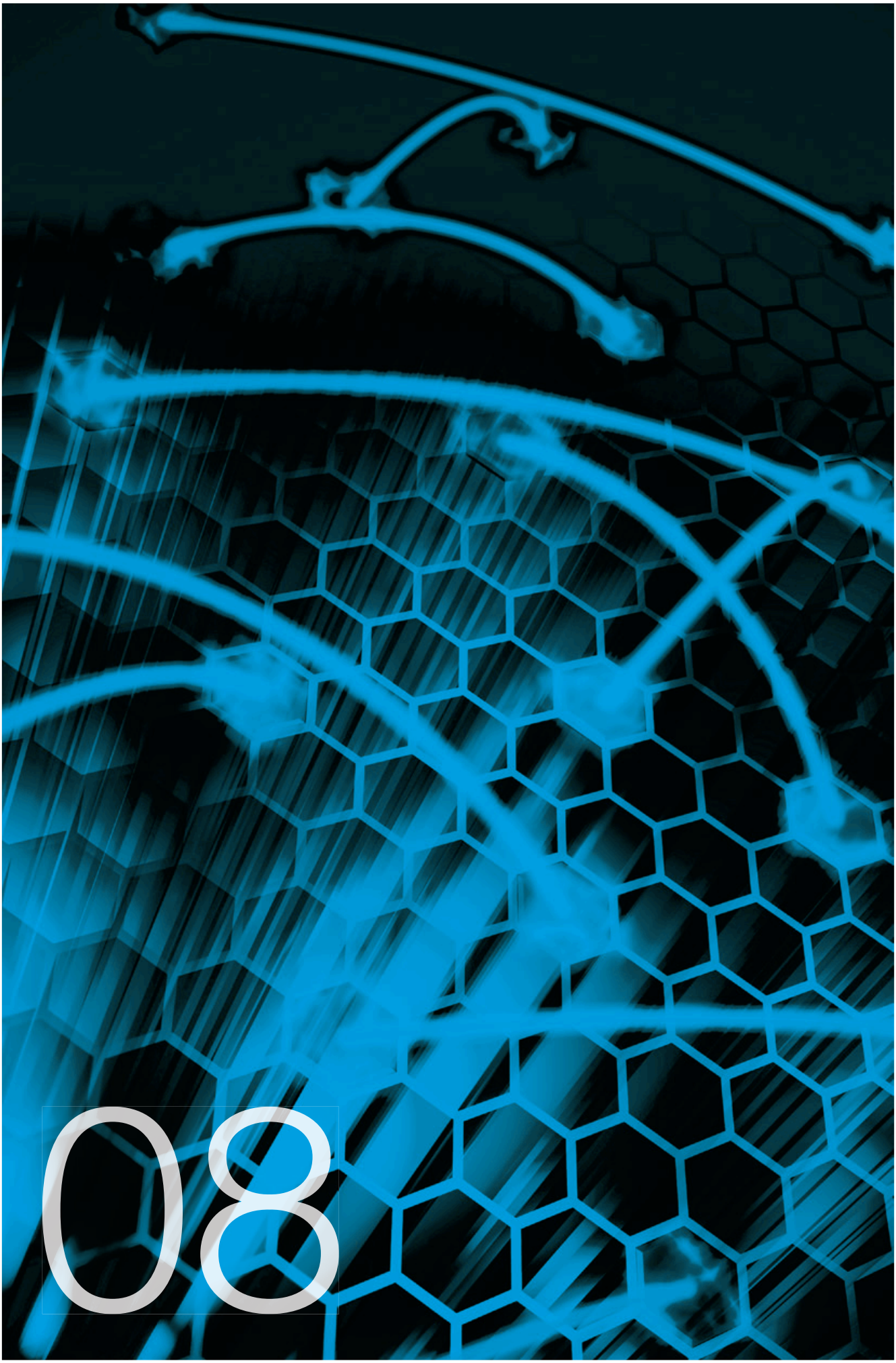
Finally, some industrial companies have completely different products in different business groups that have very limited technology leveraging among them. In this case, a decentralized software organization structure should be considered. If there is a resource constraint, they could consider initially starting with the software R&D for a few limited products.

These decisions won't necessarily be permanent; as hardware companies move from a single-minded focus on manufacturing to a broader focus on delivering end-to-end offerings, their software organizations will need to change as well. In the transition from one model to another, executives may need to introduce key performance indicators and other metrics to help the software organization (however it is structured) quantify the impact of its development efforts and to help project leaders set and meet personal targets. Because of the global scarcity of technical talent, leaders of traditional, hardware-oriented companies may need to adjust some of their human resources practices—for instance, providing attractive, high-profile assignments in which software experts actively participate in product design and planning, or letting software engineers lead higher-level strategy discussions. Most important, executives who are bringing software R&D in-house will need to become steeped in basic software terminology and concepts. They don't have to be experts, but gaining at least a rudimentary understanding of what the software can and can't do may help them achieve their business objectives in the long run.

□ □ □

The software development function in many hardware-focused companies flies under the radar—until growth slows and executives with cost cutting in mind notice the large cadres of engineers they've acquired over the years or until a new business opportunity emerges and executives notice how few engineers they have on staff. Executives need to be more proactive; they need to recognize the complexity and collaboration associated with software development and react accordingly.

*The authors would like to thank Harald Bauer and Ondrej Burkacky for their contributions to this article.*


08

# Integrated expertise: Models to break siloed software development

Santiago Comella-Dorda, Chandra Gnanasambandam, Bhavik Shah, Tobias Strålin

The tradition of completely separate organizational functions is incompatible with effective software development. Understanding the options for functional integration and embedding knowledge across units can help deliver substantial value for software organizations.

Software organizations have historically been divided into functional groups with developers, quality assurance (QA), user experience (UX), security, analytics, and operations sitting in distinct functions. With the exception of a growing number of cases in which QA teams are more broadly integrated, the software organization's functions are walled off from each other and operate independently. As software makes the transition from an on-premise model to the cloud, however, this siloed way of working is being challenged.

## Cloud's software disruption

The shift to cloud is driving three distinct software development changes. First, vendor operating models are changing. Operational responsibility now lies with vendors. This places greater importance on reliability, uptime, and the operational effects of the software architecture and the resulting application. Efficiency, scalability, and performance of the application are also becoming more important because more resources are required to support the application. Operational responsibility means a greater security burden, making vendors more active in safeguarding customer data. The shift to cloud also gives vendors access to new capabilities—including advanced analytics, A/B testing, fine-grained customer segmentation, and continuous deployment—making it easier for them to handle the demands of their new responsibilities.
The scalability and reliability requirements of cloud applications have also forced vendors to dramatically improve how they modularize, deploy, and monitor the application. "Hot deployment," horizontal scalability, and transaction monitoring are fundamental vendor capabilities, requiring a diverse set of functions (development, release management, performance engineering, UX, security, for example) to collaborate effectively.

Second, cloud enables iterative and agile development methodologies along with continuous deployment. These methodologies have shortened the development cycle and the expected time to market.

Third is the fact that performance differences have narrowed across emerging platforms. This convergence has led to the increased importance of design and UX as differentiators. The focus on user experience can no longer be just about placing the button at the right location on the Web page or selecting an appropriate icon. There has to be end-to-end consideration that ensures the application experience is both intuitive and innovative. Together, these cloud-driven trends are making the case for sharing expertise across functions and integrating it into development teams.

The traditional siloed software development approach not only slowed down the overall development cycle, but it also produced low-quality products. Integrating functional expertise into development teams can facilitate major improvements in the development of both cloud-based and on-premise software. McKinsey's software productivity benchmarking data shows that a collaborative, cross-functional approach reduces schedule slip by 30 percent, cuts down residual defects by 70 percent, and improves overall productivity by 27 percent.

## Archetypes of functional integration

The move to cloud is making old ways of working obsolete, and software organizations are responding. They are beginning to integrate the knowledge of the QA, UX, security, analytics, and operations groups into development teams. Integrating these different skill sets and perspectives is giving software organizations the added power they need to perform in a rapidly evolving software environment. In a survey of software organizations, McKinsey identified four, distinct organizational approaches to integrated expertise.

Fully embedded resources. With this approach, individuals with expertise in various functions become wholly a part of the development team, and those experts are part of the standard team configuration. They report up through R&D, and priorities are set entirely by the R&D teams. This model is best suited for teams or business units that operate mostly independently and when consistency across teams is not required. It has the potential to create the greatest agility within a development team. It also gives the experts the best opportunity to understand the development team's function and become familiar with the application. UX and operations tend to be the teams that most commonly fully embed their resources into development teams. The trade-off of this decentralized model is that it creates quality inconsistencies and design fragmentation across product groups. To mitigate this, companies that adopt this model create "communities of practice" across embedded teams to share best practices and align on approaches up front. Also, this model tends to lack the structure to support the growth of functional experts within their areas, limiting their professional development opportunities.

Semi-embedded resources. In this model, experts also work full-time within a development team, but they report through a separate organizational hierarchy. Experts across teams collaborate and establish and drive new standards for the organization. This model is as applicable for the UX and operations teams as the fully embedded model, and companies actually find this one more scalable. The software organizations that participated in the McKinsey survey reported that this model was suited for the security and data analytics roles too. The benefits of knowledge absorption, in general, apply to this model as well. The central governance aspect of this model also promotes a level of consistency. Development teams with integrated operations expertise—known as DevOps—enable experts to better optimize performance or respond more effectively to an incident with the knowledge of the application. Integrating UX expertise can result in a consistent experience for users across all product groups. This is especially important if products across the organization need to work together seamlessly and be integrated. This approach also enables consistent training and skills development across the organization. Some R&D teams, however, have reported that some applications present a rather steep learning curve for those joining the product team. For UX and security experts, for example, the idea of "just-in-time" resource sharing may be impractical.

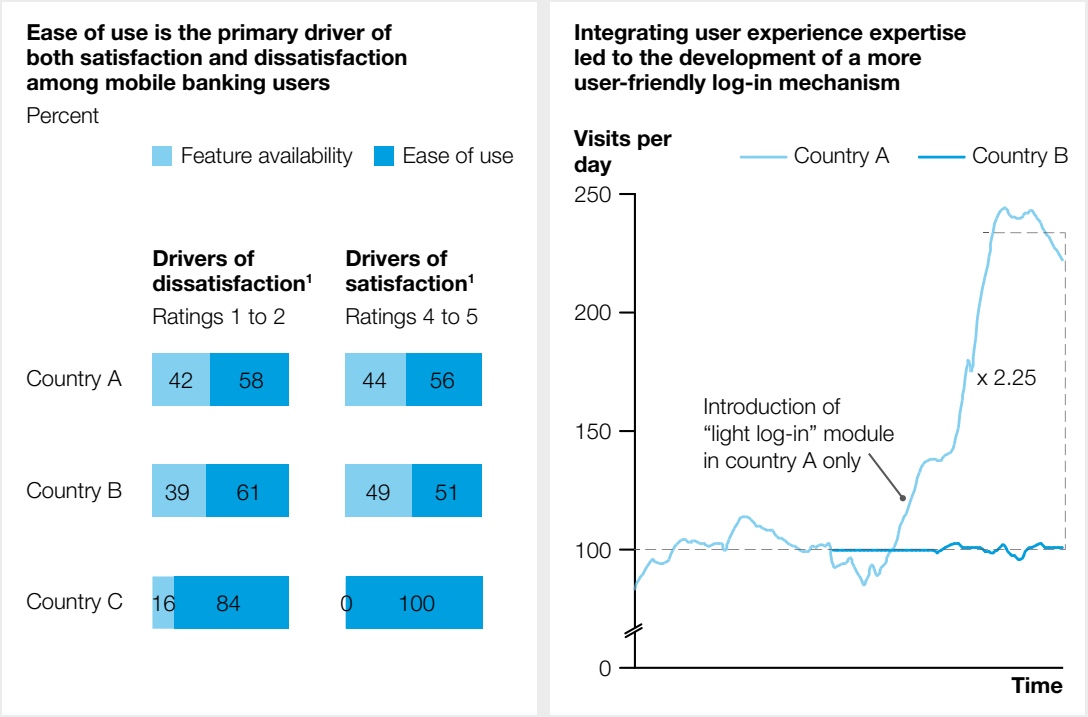
Centralized expertise. In this model, the functional expertise for development teams comes from a centralized external team that provides core capabilities to all development teams. Typically, the central team is comprised of senior-level functional experts who create guidelines and standards for all teams to follow. This type of model works very well for complex topics like security. The capacity for application teams to develop products informed by experts in other functions is built slowly. This model is less radical than the first two in that it doesn't break down the siloes. It is, however, a suitable option for integrating specific expertise not required at scale. It is equally suitable for small and large organizations.

Developer-owned/rotational expertise. This model requires development teams to dig in and take ownership of areas that lie outside

**Exhibit 01**

**Embedding user experience expertise helped one bank dramatically improve customer satisfaction**

Monthly log-ins per unique user (disguised example), indexed




[1] Based on sample of app reviews, respective rating and theme of the comment, i.e., a review rating the app with 1 star and commenting on the hassle of log-in counts as dissatisfaction and ease of use

Source: McKinsey analysis

of their expertise. Without expert guidance from the embedded models or consultation from the centralized model, engineers are expected to fulfill the cross-functional duties themselves. This model requires development team members to take on various expert roles for a predefined period of time, then rotate responsibility. Given the scaling difficulty, this model may be most appropriate for integrating operations expertise into the development units within start-up or start-up-like environments. It encourages the dissemination of knowledge across the organization, but it can create the most fragmentation. Teams also lose the benefit of having dedicated expertise, since developers are regularly out of practice between their rotations. Smaller companies reported that the pain and benefit of this model was that developers quickly understood the difficulties other functions faced and worked with them to identify appropriate solutions either by changing how they designed and coded applications or collaborating better.

## Implementation and impact

The degree to which individual engineers develop deep functional knowledge varies by archetype. Regardless of the model chosen, however, there is a shift in the expectation of all software engineers. It is important that all developers possess security knowledge, for example, and apply its principles when architecting and designing products. Organizations report that implementing one of these models to integrate security expertise into the development unit has promoted a "culture of security."

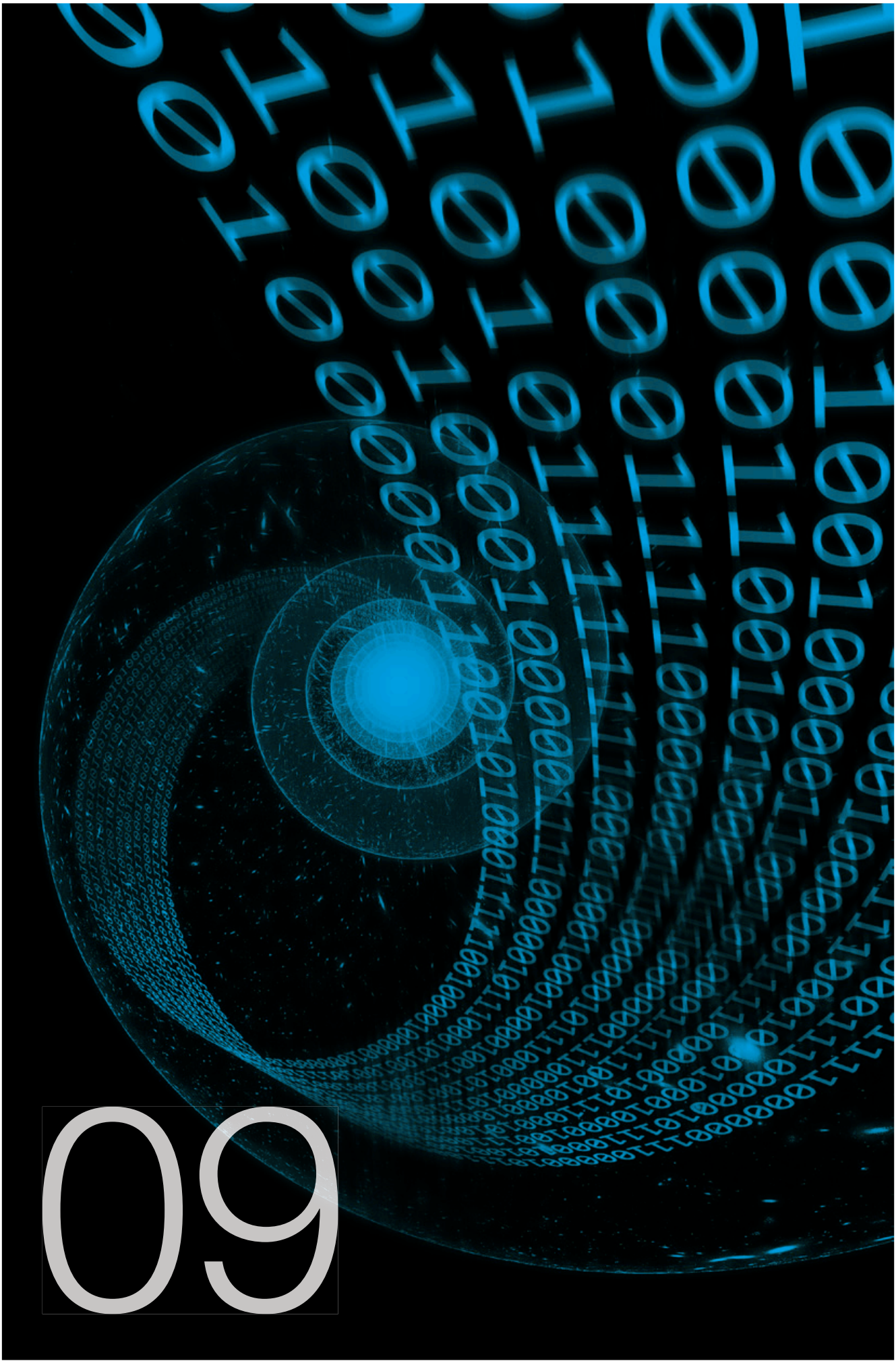
Software organizations looking to integrate functional expertise into their development units don't have to select just one of the archetypes described above. Integrating one type of expertise into a development team may require one approach, while another expertise may be best suited to a different model. Each function can operate in a different archetype, depending on the function's capabilities, scale, and role within the broader organization's priorities. Organizations have the option of implementing one or a combination of these archetypes (for example, UX can be fully embedded, while analytics is semi-embedded). While organizations have implementation flexibility, McKinsey's survey found that the "fully embedded resources" and "semi-embedded resources" models are the most common for the integration of all four functional roles—UX, security, operations, and QA.

Successfully integrating functional expertise into development teams can deliver substantial value to a company. Looking at UX as an example, organizations have shown improvement in conversion and customer engagement by doing this. One bank, in particular, analyzed feedback on its mobile banking application and noted ease of use was frequently cited as an issue. In response, the bank fully embedded UX resources. It then performed a controlled rollout of the new functionality in one country as an A/B test. After monitoring performance in that country and seeing increased frequency of usage after launch, the change was rolled out to other countries. Fully integrating UX expertise into teams led to improvements in the mobile banking application and an increase in user engagement of 2.25 times (Exhibit 1). The model for integrating UX expertise was subsequently adopted by all major product groups.

Evidence also exists showing the potential of DevOps teams. Research reveals improvements in asset utilization of over 25 percent and significant decreases in provisioning time. According to a recent study by DevOps software provider Puppet Labs, teams that integrate operations and commit to continuous release practices deploy code 30 times more frequently, have half the number of production failures, and can restore services 12 times faster after a production issue.

□ □ □

Software's rapid evolution from on-premise toward cloud-based platforms is enabling, among other advances, continuous deployment. With this opportunity, however, comes the need for greater agility within software organizations. The traditional ways of working that isolate developers from the functions of UX, security, operations, and QA no longer suffice. Organizations must integrate these elements of functional expertise into their development teams. Their options for doing so comprise varying degrees of resource dedication and different organizational structures. They all, however, require developers to understand and apply functional principles in ways that haven't been required before. Those that successfully integrate this expertise are seeing improvements in productivity, quality, and user engagement.



# 09

# Quality code: Driving successful software development

Peter Andén, Tobias Strålin

As software becomes ubiquitous, companies continue to struggle with development quality issues. A comprehensive approach to development quality can rapidly produce tangible improvements.

The world runs on software systems, which power everything from cars to consumer electronics and from smartphones to intelligent appliances. Yet as more industries cede increasing amounts of their products' functionality to software, the stakes are becoming increasingly higher. At one time, problems associated with code development meant little more than an error in a stand-alone system resulting in customer frustration. Today, the aviation, automotive, and financial services industries among others, rely heavily on software code, and poor quality can have disastrous consequences, threatening public safety and global economic stability. One aerospace company, for example, ended up grounding an entire aircraft fleet due to software issues, and a global premium automaker had to recall nearly a million cars due to a safety-related software glitch. A personal computer manufacturer ended up shipping PCs with a virus already resident in the installed software, while a bug in trading software cost a firm nearly $450 million in 45 minutes.

The problem will likely get worse before it gets better, driven by the trend toward more complex interconnected systems. For businesses that increasingly rely on software systems, the quality of the code itself can be a key differentiator for customers, tracking closely with repurchase intention and satisfaction levels.

McKinsey has conducted significant research into software development, reviewing more than 1,300 completed software projects across industries and conducting a large number of interviews with software managers, architects, and test leaders. From this research, it is clear that best-in-class software players are twice as effective as average performers at producing high-quality code, which has become a strategic skill in the current environment.

Based on this experience, McKinsey has formulated ten core beliefs regarding software quality that cover strategy and governance, development and testing, and capabilities and mindset (Exhibit 1). These core beliefs can help companies create and maintain an organizational culture that prioritizes and builds high-quality software.

Once the organization has set the ambition to improve its software development quality, the management team needs to launch a number of initiatives covering the full development cycle, from requirements collection to customer acceptance.

## Specifying scope, architecture, and planning

When establishing the scope of the software development project, high-performing companies standardize the intake process for setting business requirements. Less capable organizations often rely on ad hoc processes, allowing multiple stakeholders to write business requirements, while development or business analysis teams refine them. The quality assurance (QA) department often does not actively participate in these early stages of development, but experience shows that their involvement is critical. In fact, a cross-functional team—including key business stakeholders such as marketing, development, and QA—is ideal in prioritizing and managing the software development requirements. In this model, requirements are reviewed from the start from a quality and testability perspective to determine if

**Exhibit 01**

**Ten beliefs in three categories underlie successful software development**

| | Core beliefs |
|---|---|
| **Strategy and governance** | 1 Strategy is based on a deep understanding of "voice of the customer" and "total cost of quality" |
| | 2 Quality KPIs should follow the strategy and be clearly defined and cascaded across the organization |
| | 3 Complexity is managed by striving toward a single high-quality main branch with daily updates |
| **Development and testing** | 4 Quality assurance is integral to each step of the software development process |
| | 5 Invest in writing good code from the start: for example, by implementing code guidelines and reviews and leveraging peer programming to build collective responsibility |
| | 6 Testing should be implemented as early as possible, leveraging test-driven development and continuous integration |
| | 7 An interlined and automated tool chain drives efficient testing and issue resolution |
| | 8 Managing software supplier quality across all parts of the life cycle and integrating it into the software organization's own quality assurance flow is critical |
| **Capabilities and mindset** | 9 A capable testing organization with leadership in senior management is essential |
| | 10 A "first time right" attitude and culture that reflect the importance of software quality must be cultivated |

Source: McKinsey analysis

the desired features are realistic to implement and to establish the testing approach and initial test cases early on.

To respond to customer requirements, it is still important to be agile and flexible in the intake, but this needs to happen within the context of a well-conceived structure. It is important to have one project-wide requirements list that is continuously updated and shared, but still owned by one responsible executive. Advanced requirements are broken down into manageable, incremental modules or work packages. There are many ways to manage how to prioritize the work ahead, and the ideal approach depends on the company's context. For a small mobile app designer, for instance, it can be perfectly sufficient to just hold frequent team prioritization meetings. For large and more advanced software releases and projects, an empowered and highly capable planning and integration team might be advisable to maintain an up-to-date feature list and integration plan. The integration plan will inform the prioritization of modules or work packages for development and integration into the main branch. The integration plan is continuously updated based on input from the project-wide requirements list—for example, new features and changes in priorities. It addresses dependencies between different modules or work packages so that elements needed by other modules or work packages are developed first.

The software architecture team should work in close collaboration with the integration team and aim to establish a modular stack with as few dependencies between the layers and modules as

possible. It is also recommended to allocate time for refactoring (architecture cleanup) during the software project to keep the software architecture as modular as possible. For projects with fundamentally new hardware, software platform, and/or architecture, it is critical to invest enough time into software architecture and integration planning up front—this, to develop an initial overview of interfaces and dependencies between different modules and work packages.

It is also critical to establish a robust process to track and manage software supplier quality across all phases of the vendor life cycle, and it begins with supplier selection. Ensuring early alignment, integrated test processes, and unambiguous relationships between stakeholders enables companies to minimize the risk of excessive change management actions. It is advised that companies only outsource certain specified software modules with clear interfaces and limited dependency on other modules and work packages.

## Implementing quality-focused development methods

Once the project has been scoped and structured, a number of initiatives should be launched to promote better quality during the development phase.

Test-driven development is an approach in which programmers write the tests to which the code will be subject before starting to develop the code. This approach helps foster a quality mindset and promotes deep understanding of the customer use cases, before beginning with the actual code development.

Pair programming brings two developers together to produce all code. One takes the primary code-writing role, while the other reviews each line of code and creates unit tests. The two typically change roles frequently—after a few hours, for example. In addition, superior software developers often adopt coding standards that allow everyone to read each other's code, and establish collective code ownership, where any team member can work on any part of the code.

Continuous integration is another very effective strategy, where teams integrate the latest version of code as often as possible—for example, every few hours or on a daily basis. This approach significantly improves quality, since issues are identified early and can be addressed immediately by the responsible party. Companies using the traditional waterfall methodology wait until the end of the development phase and then do one large-scale system integration. This approach, however, makes identifying individual problems difficult and requires much more time and effort to fix them. Continuous integration also shortens the development cycle—in one case, a company used it to cut its development cycle in half.

Refactoring is a development method in which the code is rewritten. By improving and modularizing code, companies can continuously improve software quality. Doing so promotes better architecture, making it easier for developers to add or change software functionality. Other benefits include the ability to identify and fix bugs in less time, as well as higher productivity, lower development costs, and reduced time-to-market performance.

The interlinked tool chain integrates tools for requirements tracking and project planning, coding, and verification. For many companies, investments in a modern tool and development environment have significantly increased both quality and productivity by, for example, reducing the time and coordination needed to fix identified software bugs and allowing significantly more bugs to be fixed.

## Succeeding in software testing and integration

After a development method or methods have been established and development is under way, companies will want to set themselves up for success in testing and integration. Focusing on the areas of testing talent, timeline, and technology can help give companies the best outcome:

Clear tester roles. To establish effective testing and integration, companies need to specify clear roles and responsibilities for the different types of testing required. This is important to ensure both sufficient test coverage and eliminate duplicated effort. It is also advisable to establish a capable test organization with clear career paths for testers—meaning opportunities to move on to more advanced roles such as test architects, test coordinators, or line management. Increasingly common today is for companies to move their top software development talent into testing organizations and shift their roles toward helping development teams write high-quality code—significantly increasing the status and influence of the testers.

Early initiation. It is also critical to initiate testing as early as possible in the development cycle, when the costs to fix problems are significantly lower than they later become (see text box "From last place to top-ranked"). Testing should be part of the software development from the start and run continuously throughout the entire development cycle.

Virtualization. Extensive use of virtualization techniques during the earliest development stages can accelerate testing and reduce costs per defect. In fact, today's technology can allow companies to test the full hardware/software stack virtually—before a hardware prototype is available. In one situation, a company noticed the increasing dependencies between software and hardware. It saw that it could test many issues only in a full system environment. Thus, it needed to build virtual prototypes to simulate the software/hardware interactions. Instead of adhering to the traditional sequence of hardware development followed by software and system-level integration and validation, the company used virtualization to do all three steps concurrently. This enabled it to reduce time to market by about 40 percent.

Additional levers to optimize testing efficiency include introducing automation, establishing a clearly defined scope of test coverage, and balancing the test-and-build cycle lengths. Automation, for example, can help companies reduce costs and enhance the replicability of tests, while optimized test-and-build cycles can make testing more effective and efficient.

## Establishing documentation and feedback

Adequately documenting test cases and results and providing usable feedback to developers is needed to reduce bugs in the software and capture insights for continuous improvement. One company found that up to 50 percent of its test issue reports were invalid for a variety of reasons. They were duplicates, for example, or testers had already fixed the issue before publishing the report. This situation can eat up significant development resources, tends to frustrate development teams, and results in poor issue tracking. To resolve the issue, the company introduced lean process design approaches, robust tracking and follow-up techniques, and special tools that improved its ability to identify and remove duplicate reports.

Publishing test reports regularly across all potential project sites—covering both test progress and error situations—can help companies improve their test validity rates. Reporting should also be automated to the greatest extent possible, integrating the reporting system with automated test protocols. Beyond this, implementing feedback loops will help companies improve their test coverage, secure results, and capture insights. In general, each prioritized issue should not only lead to a fix, but also to a discussion on how such an issue can be prevented from reoccurring—should design guidelines be updated, developers trained, or additional test cases introduced in earlier test phases?

Another quality-focused technique companies can use involves storing all requirements and test cases in a test repository. By logging detailed information in the repository, the company enables teams to backtrack errors effectively. This approach cuts the cost and effort associated

### From last place to top-ranked: An electronics player's path toward software excellence

A leading global consumer electronics manufacturer with over 2,000 software developers had a history of poor quality management. It suffered from a large number of known defects and lacked adequate product development transparency and key performance indicators (KPIs). The company decided to launch a transformation focused on software development quality. It started by diagnosing the maturity of its quality system, which covered four specific areas: strategy and KPIs, assurance and testing, capabilities and mindset, and quality organization and governance. The manufacturer also identified significant improvement opportunities across the end-to-end quality system, including fostering the capabilities of certain development teams and increasing the effectiveness of its testing organization. The diagnostic was complemented by a quantitative benchmarking effort to identify an ambitious and realistic improvement target. The manufacturer then designed a "future state" software development organization and established a multiyear transformation program to deliver on its ambitious objectives. It ultimately launched more than twenty cross-functional efforts covering the end-to-end value chain.

The company implemented the best practices described in this article and summarized the associated core principles to improve software quality as the following:

**Early issue identification.** Teams resolve issues where they occur and work to reduce issue inflow. They might, for example, use design review virtualization to detect and fix faults early.

**Global and holistic approach.** Taking advantage of the company's global scope, teams identify and standardize best practices and seek strong cross-functional involvement by tying in all necessary stakeholders.

**Learning organization.** They create a better problem solving and issue correction process throughout the product life cycle and that facilitates continuous improvement.

Because of the transformation, the company experienced tangible, verified improvements in software quality—and they went from worst to best quality in the market. Many of the underlying software quality drivers also showed fundamental improvements. The introduction of new software stability testing tools, for example, helped it uncover 250 percent more stability defects, and the use of root-cause analysis enabled it to boost its "fix rate" of identified issues from 20 to 80 percent while cutting the lead time it required to resolve issues by 80 percent.

with testing by making it possible for teams to reuse test cases, promoting easier calculation and generation of test metrics, and increasing testing effectiveness.

## Using quality metrics to manage performance

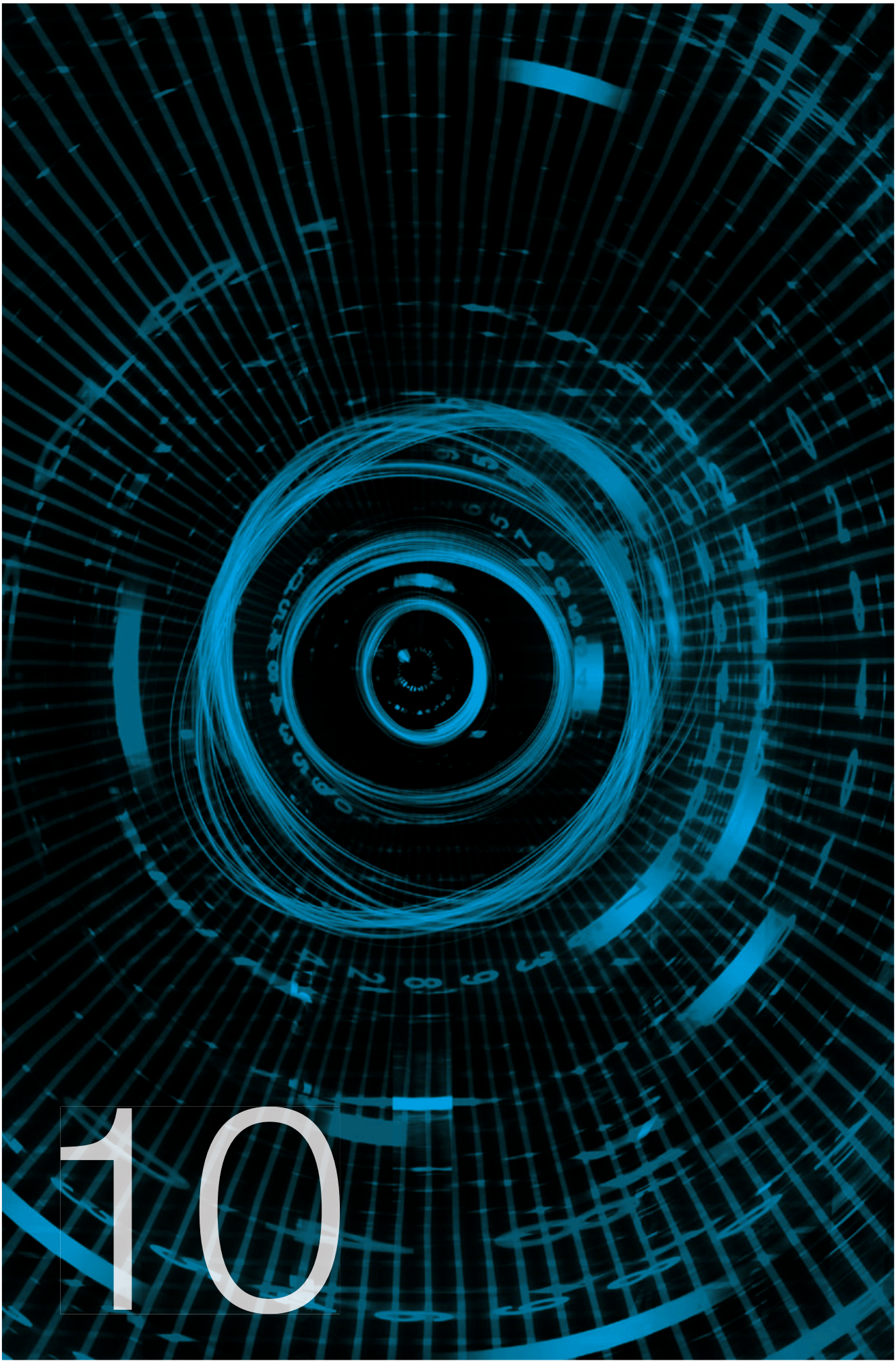
Software development organizations and teams should define their quality key performance indicators based on four considerations. First, they will need to develop a clear profile of the external customer whose preferences they will address and whose voice they will represent. Next, the organization can specify functional and nonfunctional requirements to capture customer quality—performance and stability, for example. Then they will select appropriate quality sensors and determine the timing and frequency of deployment. Finally, organizations

should establish a process to deal with both upstream and downstream results and drive continuous improvement—for example, by ensuring that defects are found as early as possible and not leaked into later development stages or even into the market.

To aid in performance management, organizations can cascade their external KPIs into internal ones and bolster the process with frequent performance reviews. With a cascaded information gathering system, the organization uses the same templates for all project subteams and relies on project managers to aggregate the subteam reports. The system calls for clear ownership of the KPIs and their targets and should provide clear processes for problem escalation, for performance reporting, and for setting targets and expectations. The company should also assign specific threshold values to each KPI and set project milestones to track performance.

As software takes control of more of the world's functionality, achieving higher quality levels has become a major competitive challenge in everything from fighter planes to luxury cars. A multistep approach to ensuring software quality that begins with the initial project scoping and ends with ongoing performance management can help companies increase platform stability, identify defects, reduce the number of customer returns, and achieve a more efficient time to market.

*The article was first published by McKinsey's Telecom, Media, and High Tech Extranet.*


10

# Continuous improvement: Three elements of managing performance

Hannes Erntell, Tobias Härlin, Tobias Strålin

Successful software development relies on the ability to continuously manage performance. By optimizing their performance management systems, companies can move beyond one-off performance improvements and look forward to sustainable bottom-line gains.

A performance improvement initiative can lead a company to greater software development productivity, increased development throughput, and higher software quality. Sustaining these improvements, however, requires something more: a performance management system. Unlike an improvement initiative, a performance management system supports continuous improvement to software development by closely monitoring software development operations. Specifically, the system focuses on flow efficiency, enables full operations transparency, and drives continuous learning. McKinsey research shows that companies with good performance management systems in place are 2.7 times more likely to achieve above-average EBITA results. Performance management is a comprehensive system that addresses three distinct elements of software development operations: metrics, targets, and infrastructure.

## Performance metrics

Key performance indicators (KPIs) are central to successfully managing software development performance because they help companies gauge overall high-level performance. The huge number of indicators across a range of performance dimensions, however, can be unwieldy. Cost, competence, speed, and productivity are just a few of at least a dozen KPIs, so companies will want to identify the five or so KPIs that most closely align with their areas of greatest concern and are relevant to all of the software development units.

The selected KPIs should be fully standardized, lagging metrics that are indicative of long-term performance. Some companies looking to build performance management systems have fallen into the trap of selecting suboptimal KPIs. Using the number of lines of code, for example, as a proxy for productivity can encourage the development of more complex or duplicative code. KPIs related to the number of features can also drive the wrong behavior, incentivizing teams to make features smaller and smaller to create the illusion of productivity. The selected KPIs should be measured either by release or by month.

The standardized KPIs should then be complemented with more tailored performance indicators to help companies understand why their KPIs change and identify corrective actions. Software development units have radically different working models: agile versus waterfall, small versus large, on-premise versus cloud-based. These additional indicators address the unique needs and dynamics of individual teams and the company's various locations, and software teams should have full flexibility in selecting these indicators.

Finally, the KPIs are also complemented with change initiative metrics to help companies track the progress of their specific initiatives. Four to six indicators and metrics per KPI—this time, a mix of leading and lagging metrics—are typically sufficient and should be measured as often as weekly.

## Business targets

Aspiration is key to performance management, and target setting is its building block. There is more than one way to arrive at the targets that guide performance management, and each

company will need to assess which approach is the most appropriate.

Team-set targets. With this bottom-up approach, teams set targets, and interventions from management are not required. Team-set targets may make sense when teams have a track record of making solid improvements on their own and within their focus areas. The transparent communication of metrics is a prerequisite for this approach. A root-cause analysis should be conducted if problems occur.

Top-down targets. Another option for setting targets starts with management. They define targets and apply them across the organization. Management introduces an incentive system to further drive improvement in selected areas. While the target is set at the top, successful implementation of this approach requires wide-scale buy-in throughout the organization.

No targets. It is also possible that not setting targets at all is a company's wisest move. This choice can allow a company to focus on turning around its organizational culture and concentrate on metric-based performance improvement without the potential confusion and added pressure that targets can bring.

## Improvement infrastructure

Companies certainly need clarity regarding where they want to go and on how to gauge their progress toward that goal. A third element—the improvement infrastructure—ensures that corrective action can and will be taken when the first two elements reveal the need for redirection. The improvement structure comprises the set of institutional mechanisms that facilitate the improvements that have been identified and enable units to continuously learn and grow.

Culture. Many organizations are beset with a culture that resists reflection and critique. Shifting the organization from a culture of defensiveness and a lack of accountability to one where assessment and continuous improvement are valued by everyone takes work. Organizations characterized by a performance-oriented culture have two things going for them. First, teams understand and are committed to the process because the process has been made transparent. Second, they have examples of a successful process because leadership models it.

Coaching. The role of coaching is to drive the dialogue around improvement in general and specific corrective actions in particular. It focuses on performance opportunities and key capabilities and creates transparency on performance-related issues. The function of coaching is also to clearly establish ownership for corrective actions and follow up to ensure that those actions are being taken.

Forums. Constructive dialogue is crucial to performance management. That dialogue is facilitated when structured spaces, or forums, exist for those conversations to take place. From lead time to quality, there are many dimensions of software development whose performance must be managed, and each of these dimensions demands its own dialogue forum. These forums adhere to tight agendas that address the root causes of performance issues and end with a consensus on next steps. The structure of these forums also includes the committed participation of the most relevant staff, and they occur regularly and frequently, typically monthly or every two weeks.

□ □ □

A company's software development function is a mix of several units and many moving parts. Improvement programs get the optimization ball rolling, but it takes a performance management system to sustain those benefits. Companies looking to implement performance management should establish clear targets, determine which metrics will gauge progress, and create an infrastructure that encourages and enables continuous improvement. Those that do so successfully will most likely show above-average EBITA performance.

# Contributors


Hannes Erntell
Principal, Stockholm

Simone Ferraresi
Associate principal, Rome

Tobias Härlin
Associate principal, Stockholm

Jörn Kupferschmidt
Consultant, Berlin

Gang Liang
Senior expert, Boston

André Rocha
Associate principal, Munich

Bhavik Shah
Expert, Silicon Valley

Christopher Thomas
Principal, Beijing

Dominik Wee
Principal, Munich

Bill Wiseman
Director, Taipei

# Authors

Tobias Strålin is a partner in McKinsey's Stockholm office. He is the founder and a leader of McKinsey's Global Software Development Practice and primarily serves leading software and high-tech players. He has helped many advanced software development organizations fundamentally transform their software development operations and make significant performance improvements.

Phone: +46 8 700 6476 • tobias_stralin@mckinsey.com

Chandra Gnanasambandam is a partner in McKinsey's Silicon Valley office and a leader in McKinsey's Software Practice. Chandra serves many of the world's leading software and services companies and has extensive hands-on software development experience.

Phone: +1 650 842 5662 • chandra_gnanasambandam@mckinsey.com

Peter Andén is a partner in McKinsey's Stockholm office and the co-leader of McKinsey's European Software Development Practice. Peter has significant experience helping high-tech and advanced industry clients address strategic and operational software topics.

Phone: +46 8 700 5401 • peter_anden@mckinsey.com

Santiago Comella-Dorda is a partner in McKinsey's Boston office and a leader in McKinsey's Agile and Scale Practice. Santiago has led a large number of software development and agile-at-scale transformations.

Phone: +1 617 753 2293 • santiago_comella-dorda@mckinsey.com

Ondrej Burkacky is a partner in McKinsey's Munich office and he co-leads McKinsey's European Software Development Practice. He has extensive experience serving semiconductor and other advanced industry clients in embedded software development.

Phone: +49 89 5594 9038 • ondrej_burkacky@mckinsey.com

Editing: Monika Orthey, Scott Reznik, KJ Ward

Design: Marc-Daniel Kress (metamorphosis7.com)

Base pictures for title and chapter illustrations: © fotosearch.de

Software Development Practice
January 2016

www.mckinsey.com