DIGITALLY REMASTERED

Continuous Delivery Critical to an Efficient Software Factory

The path from idea to business outcome—fueled by the need to innovate for competitive advantage—is now the critical path at the heart of your business. The software factory that makes the idea real must produce at a rate that allows you to deliver value to your customers at the speed they want to consume it; any slower means missing customer demand. Waterfall development—with its monolithic projects, long lead times and static requirements—falls down on delivering both speed and value. In stark contrast, agile methodologies decrease cycle times and increase delivered value through the continual iteration of small teams freed from the chains of bureaucratic development processes. Because that iteration continually revisits what you're building in the context of what your customer wants, you increase speed by reducing rework, and increase value by intensifying engagement around the idea.

In this "Digitally Remastered" excerpt, you'll learn the importance of Agile and show to scale it to support the largest software factories.

Building Your Factory Team

The essence of your software development capability comes from the collective power of the individuals you bring together to form your software factory team. Addressing the "need for speed" that is part of all software development cycles will require a combination of the right tools and technologies as well as people with the skills to use them effectively to architect and operate a friction-free software production line. You will need to create a highly automated system with the right people and processes to integrate, test, deploy, and operate the software you develop. In parallel, you must also acquire the capability and skills you need for "above the line" software innovation — all the pieces that make up the experience delivered to customers. This includes everything from user interface and interaction design to software architecture and development.

The following list summarizes the key considerations you need to keep in mind when building the team you need:

- **Some of the roles and titles you currently have in place may give you a false sense of security.** You must reevaluate and refresh your roles and job requirements carefully to ensure that you can retain and attract the skills needed to design, build, and operate customer-facing software experiences.
- **You may need to bring in expertise that you didn't previously have because there was no need.** Product designers, software architects, and app developers are common skills gaps in enterprises. The business of building software experiences is a radically different endeavor than building line-of-business tools, and you need to look beyond filling jobs that have been defined by the needs of classic, internally focused IT.
- **When hiring, go for depth and breadth.** There's a move today towards acquiring higher-order problem-solving acumen in combination with core math, science, or coding expertise — the "T"-shaped employee with deep domain-specific expertise who is also able to think broadly across multiple problem domains. The type of innovation you need requires creativity, lateral thinking skills, and the ability to operate with high levels of ambiguity.
- **Your factory will constantly be changing and evolving — sometimes rapidly — in response to customer need.** Identifying candidates who can embrace change and who can thrive under uncertainty is of equal importance to obtaining candidates with specific technical skills.
- **In the fast-paced software world, existing skills become outdated quickly, and the ability to change and learn is essential.** Hiring against a list of technical specs may be useful to fill a specific need right now, but your list will change rapidly over time. You must also hire for the ability to change, adapt, and most importantly, continuously learn.

As you build your software factory team, you will need to find the appropriate organizational framework to support it. You likely already have an existing technology organization with responsibilities that must continue to be addressed. Do you start a completely new group? Do you refactor or repurpose existing groups to align to a new, broader mission? Will your new software factory operate in parallel with your existing technology efforts?

How you answer these questions will depend on many factors, including business need, existing capabilities, and company support. You may be best served by starting with a smaller, more focused effort working in parallel to existing groups, or by driving sweeping changes across the entire organization, or with some hybrid approach. Regardless of what you settle on, you must have a clear picture of your ideal end state and the steps you will take to achieve it. Keep in mind that the full power of methodologies such as agile and DevOps can only be fully realized at scale when applied across the entire organization. Don't stop until your transformation to a new way of operating is fully complete.

To attract and retain the talent you need to achieve your goal, you will need to review your compensation and ensure the right roles are defined and enable collaboration in the right ways to drive value. You won't just be competing with other enterprises for your talent. You will also be competing with start-ups and software-native companies. Do you have the right environment to attract designers, architects, and developers? Do you have the appropriate physical space for teamwork and collaboration? Have you streamlined internal processes, provided the right tools, and removed friction points and roadblocks to enable your team to focus on this mission?

Common Obstacles to a Smoothly Operating Factory

In computer applications, parallel processing — working simultaneously on different aspects of a task — can definitely accelerate throughput of work, but it requires careful communication and coordination to achieve its potential benefits. This is also true when the concept is applied to your software factory. It is possible for teams to work simultaneously on different aspects of the software product to quickly build new functionality and to identify and address design flaws, security gaps, performance problems, and a host of other issues well in advance of going live. For such cooperative work to succeed, however, careful orchestration between the elements of work and the sometimes-fuzzy line demarcating development and delivery of the customer experience is required to ensure that no element of work is blocked and that everyone is moving forward in unison. This way, you can spot performance problems well before development of a feature is complete, quickly adjust the user interface during the initial stages of software design, or change the way a feature works based on early feedback.

Achieving the full benefits of parallel workflows moving together at full speed will require addressing a number of key challenges. The following list summarizes some of the obstacles you may need to overcome:

- **Lack of automation.** To achieve desired development velocity, teams must be able to deliver multiple software "builds" — new versions of a software product incorporating the most recent changes and additions — every day. Each of these builds needs to be tested and validated in order to quickly move them through the software delivery factory.

  This goal can be compromised when teams have to manually create and configure the technical environments needed, perform significant amounts of hands-on testing, or deal with unanticipated differences in how the software works as it moves through different environments — the "it worked fine on my PC" problem. This can cause delays and pressure to implement short-term workarounds that introduce further fragility in the system that results in costly rework later.

- **Poor orchestration.** In the modern software factory, autonomous teams are empowered to use their own sets of tools for areas like software change management, version control, provisioning, configuration control, and release management. Some combination of these tools often don't work together, and orchestrating all the interdependent functions in order to coordinate releases requires manual processes or scripting which can take more time than the tools are meant to save.

- **Bottlenecks and delays.** Many organizations are plagued by delays due to lengthy handoff protocols and a lack of task and activity coordination. Delays are harmful by themselves, but can also encourage other poor practices impacting quality. For example, developers must "batch up" their code changes to work around long release delays, making it much more difficult to coordinate those changes with others in the system and leading to higher likelihood of problems when it's time to integrate all of the queued-up changes.

- **Overreliance on operations.** Many organizations have placed a lion's share of responsibility on the operations team who are often called in at the end of the development cycle to ensure that applications are production-ready. Because this happens late in the release process, key insights that could have influenced major architectural, development, and system design decisions are missed. The operations and development teams must collaborate throughout the entire development lifecycle to ensure that development informs operations and vice versa. By the time low-quality code hits production, it's too late.

- **Testing silos.** Testing software requires knowing exactly what it's supposed to do. Problems can arise when separate groups within the organization create differing descriptions for how software functions should work, leading to conflicting and inconsistent interpretations of correctness. Those differences are then multiplied across different tools and tests. A core part of the problem is the very notion that testing — assuring the quality of software produced — can be the responsibility of dedicated testing teams. Throwing code over the wall to a testing team to ensure that the code is of high quality and performs as needed is an impossible task, yet this is how many development organizations operate.

Getting Started

Accelerating the pace of your software delivery is vital for achieving a rapid flow of new value to your customers. There is no silver bullet — no one-time program that you can administer and then be done. You have to be ready to approach creating this capability just as you must approach your software experience delivery — as a process of constant iteration and improvement. These are some guidelines to apply and steps to take in building and optimizing your software factory:

- **Integrate teams.** Modern software development is a team endeavor and requires a high degree of integration between people and processes. Instead of having sharp divisions between design, development, testing, security, operations, architecture, and other disciplines, you must work to make those boundaries as permeable as possible. Everyone has a role to play and brings specific skills, but the focus must be on outcomes and adaptive problem solving. Adopting agile practices will help you ground your organization around small, cross-functional teams working together to deliver value to your customers.

- **Integrate quality.** Ensuring software quality must be the responsibility of the entire software factory from design all the way through operations. While software testing is a specific and necessary discipline, its primary focus should be to ensure that sound testing practices are followed throughout the entire factory. For example, developers should be writing unit tests for the code they develop and helping to incorporate those tests into an automated part of the software build process. User interface and interaction designers need to engage with test development to ensure that the user experience meets design objectives.

- **Automate everything.** Your customers don't care about your internal inefficiencies, manual processes, or friction points. What they care about is the software experience you deliver. Anything that gets in the way between you and what your customers want is moving you in the wrong direction. Manual processes are not only slow, fragile, and error-prone, they also represent opportunity cost — effort that could have been spent on creating new value. Of course, not everything can be automated, but everything that can should be. Automation will require upfront investment with downstream payoffs. You will need to be patient with your investments and persistent with your efforts.

- **Prioritize architecture.** In the constant rush to deliver the next set of features or functionality, it's easy to lose sight of the longer-term evolution of your software and systems. A constant short-term focus will ultimately result in a build-up of technical debt — an accumulation of technical shortcomings that over time creates a tangled web of complexity and fragility. Left to build too long, technical debt is very expensive to pay down and will limit your forward progress while you do. Instead, you must balance both near-term deliverables and long-term evolution so that your factory never becomes obsolete or mired in its own legacy.

Lockheed Martin uses DevOps to transform software development

Changing the software development process at a large company is a process in itself. For aerospace, defense, and technology giant Lockheed Martin, the adoption of next-generation methodologies like DevOps is as much about organizational and cultural transformation as it is technology. And it doesn't happen overnight.

"We've been on this evolutionary path to do more from an efficiency perspective," says Liz Michaud, Lockheed Martin's Director of Business Applications. Progress has come by focusing on the ways people work together and the skills they bring to the job, and this approach is starting to pay off. "We have had some thrilling success applying the DevOps model to some of our newer applications," she says.

When Ms. Michaud took her current job three years ago, Lockheed Martin had already begun the process of breaking down development and operational silos that in some cases went back decades. But there was more to be done to modernize the approach of her IT operations staff, which includes about 800 people who support some 1,600 internal applications across the sprawling $45 billion company. The operations team has responsibility for everything from manufacturing and finance to supply chain and human resources.

One key organizational change was moving to a more centralized IT operations structure with a view of the entire enterprise, and then making sure the IT professionals were closely aligned with the business areas they supported. For example, a senior IT manager will now work directly with a program manager on the business side, with the business team defining what it needs from IT to fit its strategy.

Getting serious about DevOps means finding the right people to do the job. Ms. Michaud identified several "pockets of excellence" on her team and assembled a core group who do what she laughingly calls "DevOpsian" work. Currently, they have about 20 programs running across the company. Cloud-based applications and automated testing are the primary targets.

Success is measured in the quality of added functionality as well as the speed at which it is produced. New metrics are being introduced for comparing the speed of DevOps to traditional baselines — but also in cultural change. There is evidence that this cultural change is happening; it is anecdotal, but compelling. One application for generating proposals proved so popular that it began spreading beyond its original user base before senior management had formally blessed such an expansion.

"People said, yeah, we're going forth with this — it is almost organically happening, and that's a telling sign about the ability to meet functional needs."