Industrial DevOps

Applying DevOps and Continuous Delivery to Significant Cyber-Physical Systems

IT REVOLUTION
DEVOPS ENTERPRISE FORUM
2018

25 NW 23rd Pl
Suite 6314
Portland, OR 97210

Industrial DevOps: Applying DevOps and Continuous Delivery
to Significant Cyber-Physical Systems

Produced in the United States of America

Cover design and interior by Devon Smith

For further information about IT Revolution, these and other publications, special discounts for bulk book purchases, or for information on booking authors for an event, please visit our website at ITRevolution.com.

PREFACE

In March of this year, we at IT Revolution once again had the pleasure of hosting leaders and experts from across the technology community at the DevOps Enterprise Forum in Portland, Oregon. The Forum's ongoing goal is to create written guidance to overcome the top obstacles facing the DevOps enterprise community.

Over the years, there has been a broad set of topics covered at the Forum, including organizational culture and change management, architecture and technical practices, metrics, integrating and achieving information security and compliance objectives, creating business cases for automated testing, organizational design, and many more. As in years past, this year's topics are relevant to the changing business dynamics we see happening across all industries and the role technology has to play within those changes.

At the Forum, as in previous years, participants self-organized into teams, working on topics that interested them. Each team narrowed their topics so that they could have a "nearly shippable" artifact by the end of the second day. Watching these teams collaborate and create their artifacts was truly amazing, and those artifacts became the core of the Forum papers you see here.

After the Forum concluded, the groups spent the next eight weeks working together to complete and refine the work they started together. The results can be found in this year's collection of Forum papers.

A special thanks goes to Jeff Gallimore, our co-host and partner and co-founder at Excella, for helping create a structure for the two days to help everyone stay focused and productive.

IT Revolution is proud to share the outcomes of the hard work, dedication, and collaboration of the amazing group of people from the 2018 DevOps Enterprise Forum. Our hope is that through these papers you will gain valuable insight into DevOps as a practice.

—Gene Kim
June 2018
Portland, Oregon

INTRODUCTION

The move to integrated "DevOps" (a concatenation of development and operations) is producing a dramatic change in the way we develop and deploy software and IT systems. The result is faster time to market, more resilient and more malleable systems, and systems that can be used to test potential innovations faster from a scalable production environment.

As DevOps continues to challenge the status quo and improve business outcomes for software systems, many of the world's larger enterprises also need to identify how to scale these practices across large, complex systems composed of hardware, firmware, and software. The ability to iterate and deploy faster allows companies to adapt to changing needs, reduce cycle time for delivery, increase value for money, improve transparency, and leverage innovations.

However, there is an industry-wide misconception that this form of rapid iteration and improved flow applies only to software or small applications and systems. In this paper, we will provide an extended definition for DevOps as it applies to large, complex cyber-physical systems, and offer some recommendations on how to effectively leverage continuous delivery and DevOps in these systems.

Problem Statement

Industry lead and cycle times for delivering significant cyber-physical solutions—systems such as robotics, warfighting, transportation, complex medical devices, and more—are insufficient to meet the increasing demands of our customers. As the size and complexity of cyber-physical systems increase, the visibility of the work items and activities in the value stream decrease, compounding the problem. Additionally, many large-solution providers are organized around functional areas and apply traditional, sequential, stage-gated development methods, resulting in multiple handoffs and delays. The net result is usually slow time to market, quality issues, cost overruns, and solutions that are not fit for their intended purpose.

Similar problems exist in the development of large-scale software systems. There, the DevOps movement—along with well-described principles, practices, and toolchains—has been shown to deliver dramatic improvements in time to market.[1]

Industrial DevOps

Industrial DevOps is the application of continuous delivery and DevOps principles to the development, manufacturing, deployment, and serviceability of significant cyber-physical systems to enable these programs to be more responsive to changing needs while reducing lead times. This practice focuses on building a continuous delivery pipeline that provides a multi-domain flow of value to the users and stakeholders of those deployed systems.

The bodies of knowledge that inform Industrial DevOps principles and practices include DevOps, Lean manufacturing, Lean product development, Lean startup, systems thinking, and scaled Agile development.

In this paper, we'll describe some guidance that can be used to leverage the learnings from DevOps and scaled Agile developments to address the challenge of building these complex cyber-physical systems in a more efficient and more effective manner. The eight recommendations (or potentially principles) elaborated upon in this paper are as follows:

1. Visualize and organize around the value stream.
2. Apply multiple horizons of planning.
3. Base decisions on objective evidence of system state and performance.
4. Architect for scale, modularity, and serviceability.
5. Iterate and reduce batch size.
6. Establish cadence and synchronization.
7. Employ "continu-ish integration."
8. Be test driven.

---

[1] *The Pentagon Wars*, dir. Richard Benjamin (United States: HBO, 1998), May 10, 2011, accessed April 28, 2018, https://www.youtube.com/watch?v=aXQ2lO3ieBA.

GUIDANCE

1. Visualize and Organize Around the Value Stream

Under the objective of efficiency, many large programs, have elected to organize around activities such as program management, systems, software, firmware, hardware, and testing. Such an organization has been shown to be quite problematic when it comes to actual delivery of the solution. Delays and economic overruns have proven to be a significant burden on taxpayers.

To deliver systems that provide desirable business outcomes, we need to organize around the value stream and the end-to-end steps that are required to deliver value, as depicted in Figure 1.

Figure 1: Value Stream: Includes All the Functions Necessary to Deliver End-to-End Value

When employing DevOps in large, complex systems, the first step is to define the value stream(s). For example, if we are building an autonomous drone to protect our troops, then traditionally our teams would be organized around functional areas such as systems engineering, hardware, software, firmware, and testing with handoffs from one area to the next. Teams organized around such activities are locally optimized and cannot deliver any end-to-end business outcome to stakeholders on a regular cadence, leading to a long cycle time for development as well as a long, costly recovery time.

To successfully build large, complex systems, we suggest the program should organize the end-to-end value stream. In the autonomous drones example, the best value stream organization would be around independent capabilities, such as a flight

camera team. To build a flight camera, we will need electrical engineering, hardware engineering, systems engineering, software development, and testing teams. These teams would remove activity-based handoffs and focus on providing the specific business outcome of providing visibility of the battleground. The team would need to have skills such as model-based systems engineering, sensor development, machine learning, and material science. Business outcomes of the product would be time-based video streams, munition detection, and knowledge of enemy locations.

2. Apply Multiple Horizons of Planning

The Agile influence in DevOps for smaller capabilities focuses on short-term planning horizons with iterations typically running no longer than a few weeks. However, large, complex systems demand longer planning cycles that are approached with the notion of multiple planning horizons. These large-scaled solutions need multiple perspectives—from a long-term view to near-term objectives iteration and program increment objectives. This is especially important in mission-critical environments where technology is moving at such a rapid pace that a solution could be obsolete before a project is completed. Large, complex solutions often require multiple planning levels in order to meet goals and ensure alignments across development teams.

As Figure 2 illustrates, the planning horizon starts with a vision to understand and communicate the outcome for the product solution. The next level for large solutions (such as fighter jets and autonomous systems) is a high-level roadmap to identify key capabilities, milestones, and dependencies that may span multiple years. The next level of planning is annual planning (i.e., what can be accomplished in the upcoming year). The annual plan is defined and broken down into quarterly time boxes to identify what deliverables can be completed in the each quarter, knowing that priorities over the course of the year may change and are frequently re-evaluated. The quarterly planning allows for many small, agile teams to visualize the work across teams and to work through dependencies between teams. These teams begin to collaborate for increased flow and feedback.

In large solutions, especially early in the development phase of hardware, flow between components often needs to be coordinated between and by the teams. Each team needs to understand the work of the other teams, as well as the interfaces and integration points. Quarterly plans are broken down into smaller time boxes known as

iterations, with the industry standard for iterations being two weeks. Sprint/iteration plans are further broken down into daily plans in which each team decides what can be accomplished each day. Each planning level provides time boundaries that enable teams to quantify "plan" versus "actual," allowing the product solution to course-correct based on empirical evidence.

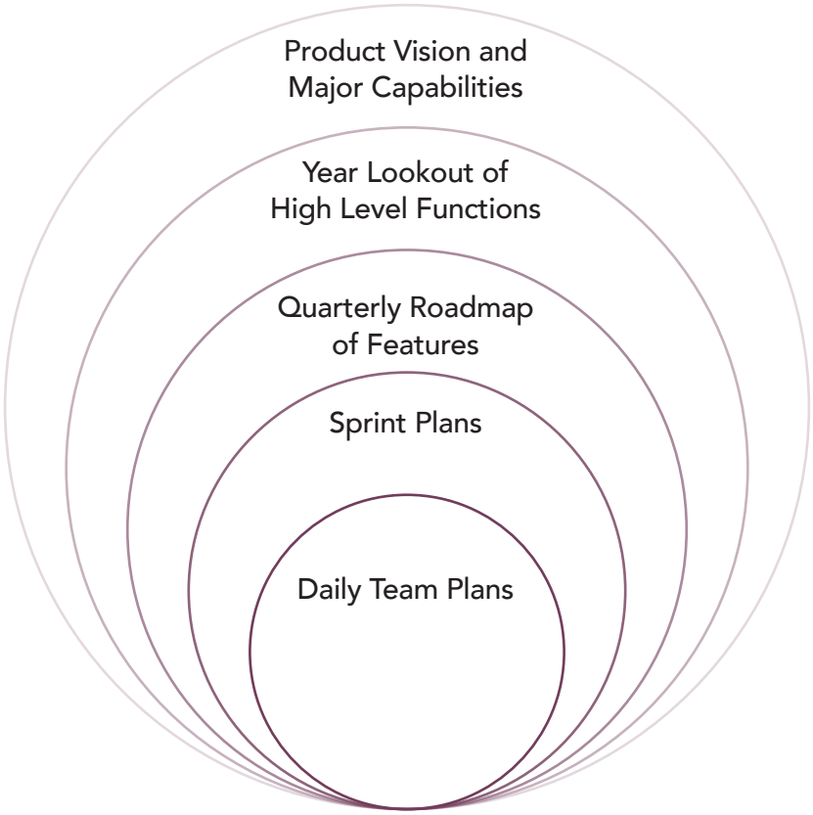

Figure 2: Multiple Planning Horizons

3. Base Decisions on Objective Evidence of System State and Performance

Throughout development, the system is built in time-boxed increments. Each increment provides an integration point for demonstrating objective evidence of the feasibility of the solution in process. That evidence is provided through a demonstration of working features of the system. For hardware and embedded systems, the early demonstrations may be limited to mathematical formulas, 3D models, walking skeletons (tiny implementations of the system that perform an end-to-end function),

or prototypes that prove a specific element of the design is viable. For example, the minimum testable feature of a new integrated circuit can start as a walking skeleton using a breadboard, as shown in Figure 3.

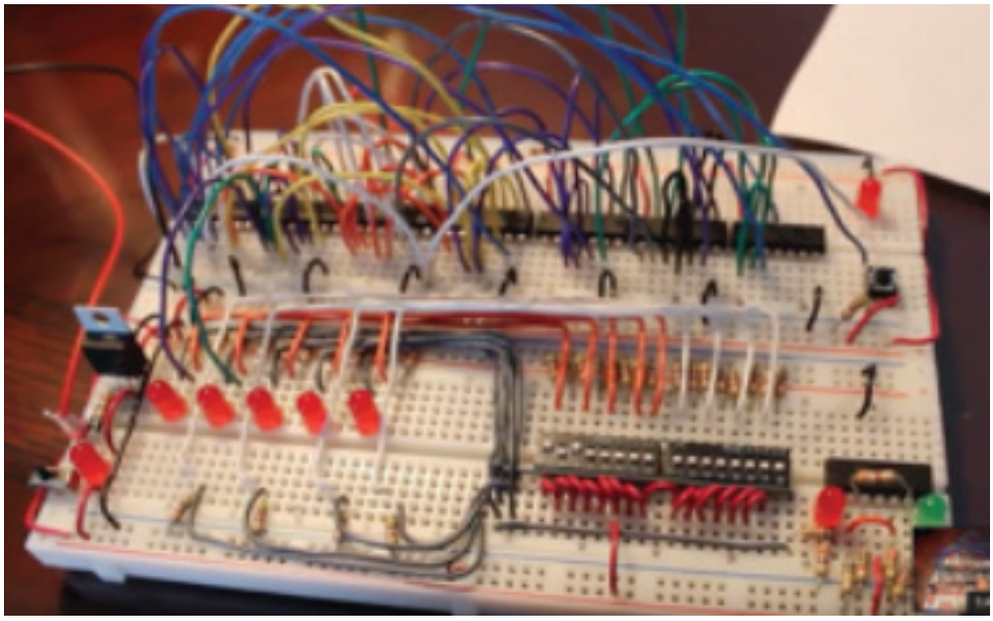

Figure 3: Example of a Walking Skeleton for an Embedded Component

Improvements are made until the final production version is fully tested and ready to deploy. Because these reviews are performed routinely on a set cadence (for example, every two weeks), system development progress can be measured, assessed, and evaluated by the relevant stakeholders frequently, predictably, and throughout the solution development life cycle. Faults can be found and corrected in small batches when the cost of change is low. The transparency of this process provides the financial, technical, and fitness-for-purpose governance needed to ensure that the continuing investment produces a commensurate return.

4. Architect for Scale, Modularity, and Serviceability

Our design and engineering processes need to scale to the level of complexity inherent in large cyber-physical systems. Architectural decisions can support this by emphasizing modularity and serviceability. Modularity refers to component decoupling with a focus on the smallest unit of functionality. Serviceability refers to a focus on lowering

the cost and time required to alter functionality both pre- and post-deployment.

Architecture modularity significantly impacts DevOps goals to continuously develop, integrate, deploy, and release value. Modular component-based architectures communicate in a consistent way through well-defined interfaces and thereby reduce dependencies between components, as displayed in Figure 4. Such components can be independently tested, released, or upgraded.

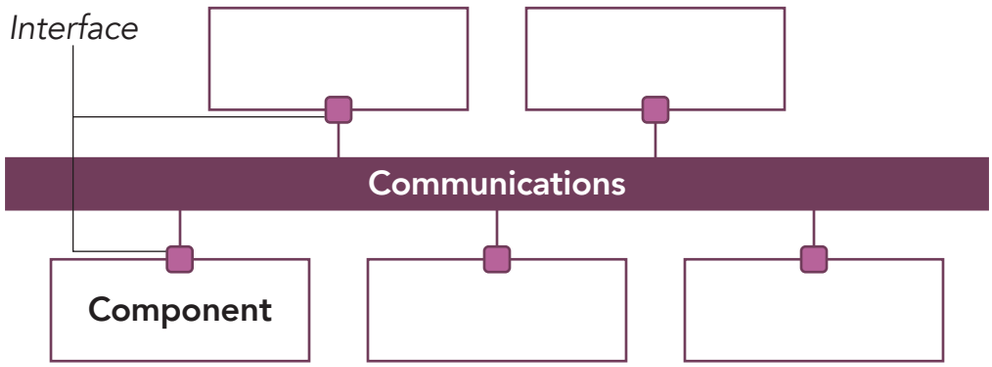

Figure 4: Components Communicate Through Well-Defined Interfaces

Architectural decisions also impact serviceability. First, there is a cost trade-off between optimizing the product's material and manufacturing costs with the products ongoing serviceability. Second, the assignment of systems functions to components and implementation choice for that function—software, programmable hardware (FPGA), customer hardware (ASIC), and ultimately mechanical parts—also impacts serviceability. For example, a common automotive dashboard has mechanical inputs wired to electrical circuits to send messages out of a common bus to control entertainment, climate, windows, and the like. Newer dashboards replace all mechanical inputs with a single touchscreen with software-defined controls, which sends messages around the system. Assigning functionality to mechanical and electrical components offers lower material costs. Assigning that functionality to software and a touchscreen provides the ability to continually release new functionality (even over-the-air) and dramatically reduces the cost of delay when releasing new value to users. In this example, releasing new value provides better economics than reducing material costs.

While some architectural decisions should be made early, many decisions can and should be delayed as more is learned about the system during development. Economics should drive the point at which exploring architectural alternatives stops and decisions become fixed. Modeling, simulation, and low-fidelity prototypes allow us to prove out architectural decisions and obtain rapid feedback as we iterate across potential solutions in a cost-efficient manner. The specific buffer between architecture and implementation will scale with the complexity of the target system and the maturity of the solution to enable sufficient time for communication and alignment.

5. Iterate and Reduce Batch Size

To enable flow, fast feedback, and continuous learning, it is important to work in small batch sizes—both in terms of the size of the component or feature and the unit of change. Small batches of functionality increase both the rate of technical exchange and the flow of work, enabling rapid feedback. Working with small components often reduces complexity and enhances transparency of the achieved results. Short iteration cycles, as displayed in Figure 5, provide stakeholders with regular reviews of results against a defined set of acceptance criteria, enable the opportunity to provide feedback, and help identify integration challenges earlier in the development life cycle. Working in short iterations throughout the development of the component or architecture enables agility and regular validation that the component is satisfying downstream expectations. Using this faster feedback model improves understanding of both the system being built and the requirements of system users. Iterative development is enabled by model-based systems engineering and well-defined interfaces. Model-based systems engineering, A/B testing, and dark launches provide the ability to make changes and rapidly understand the impact of those changes.

There are unique distinctions when developing hardware in small iterations as compared to software. Lead time associated with fabricating hardware creates a specific order on how the work for the iterations is defined and planned. This results in a more defined flow of the development activities, limiting some of the flexibility in reprioritization, yet still taking advantage of the learning and feedback opportunities each iteration brings. With hardware development, the results of an iteration may not produce something an end user can use; that is, it is not "potentially shippable" but still results in functionality and objective evidence of completed work that can be

tested against a subset of the acceptance criteria. For example, early iterations may involve 1:1 paper cut-outs of hardware that enable quick feedback from manufacturing. Later iterations can evolve to a 3D rapid prototype after an approach has been agreed upon.

Short iterations provide the opportunity for teams to plan for the earliest integration points possible and regularly validate requirements and interfaces. The goal is to find the earliest point for integration of hardware and software in a defined integrated solutions environment. Early iterations of hardware development may focus on the creation of an emulator, while later iterations focus on the creation of a first prototype, with refinement and enhancement of the prototype guided by the results of each completed iteration.

Later in the product life cycle—such as when a vehicle or aircraft is already in operation—there is the opportunity for more frequent delivery of software enhancements into the production environment.

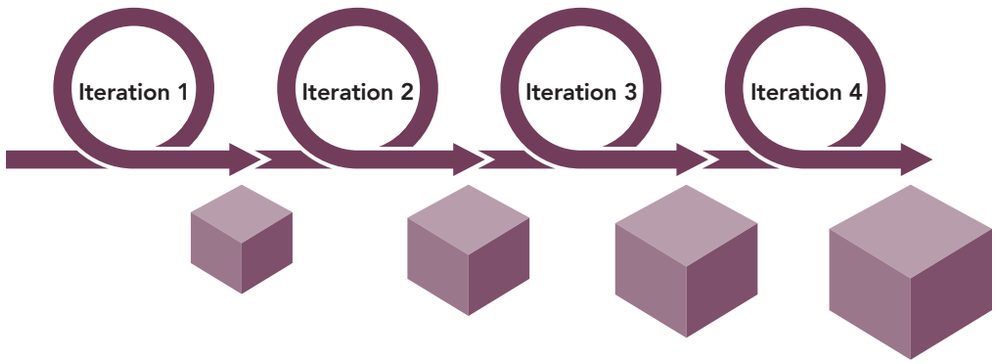

Figure 5: Frequent Feedback Loops

6. Establish Cadence and Synchronization

Applying cadence and synchronization can help manage the inherent variability in solution development. Cadence provides a rhythmic pattern—the dependable heartbeat of the development process—and provides predictable time boxes and business rhythms that allow knowledge workers to focus on solution development and variability management. Cadence also makes routine that which can be routine and makes wait times predictable, leading to lower transaction costs of key events, including planning, integration, demonstrations, feedback, and retrospectives.

However, cadence alone is not sufficient, especially where cadences may traditionally vary greatly between different disciplines. For example, hardware design, manufacturing, and testing have different, longer lead times and cycle times than software.

Synchronization between these varying cadences for significant cyber-physical systems, such as aircraft or satellites, is necessary to enable Lean flow with frequent integration, which Boeing discovered when they built the "dream liner," which started Boeing's outsourced integration strategy that resulting in large cost overruns.[2]

Synchronization allows all teams participating in the development of a large, complex system to align their efforts in time. Because all cross-discipline teams adopt the same cadence, they will also start and stop on iteration boundaries and larger program increments together. This allows multiple perspectives from the various teams to be understood, resolved, and integrated at the same time. It also enables all teams and key stakeholders to gather periodically for cross-domain planning of the next increment of development. As illustrated in Figure 6, together cadence and synchronization give system developers the tools they need to help manage the complexity and variability of large-scale solution development.

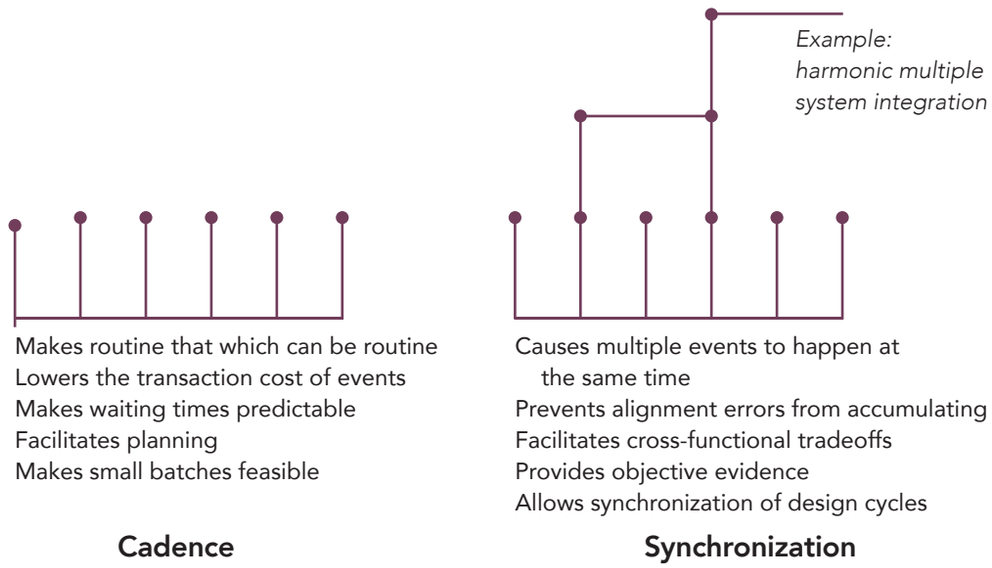

Figure 6: Cadence and Synchronization

[1] Denning, Steve. "What Went Wrong At Boeing?" *Forbes*. January 21, 2013. https://www.forbes.com/sites/stevedenning/2013/01/21/what-went-wrong-at-boeing/#17cb14207b1b.

7. Employ "Continu-ish Integration"

Continuous integration is the heart of DevOps. However, even in software-only systems this is no small feat, and teams and programs invest significant amounts of time and effort—along with purchased, open source, and custom tooling—to address the challenge.

Cyber-physical systems are far more difficult to integrate continuously, as there are limiting laws of physics as well as supplier, organizational, and test environment practicalities that must be considered. In addition, some components have long lead times to take into consideration, and you certainly can't integrate what you don't have. In its place, "continu-ish integration" is a euphemism that indicates a planned strategy to integrate frequently, based on the economic tradeoff of the transaction cost of integration versus the risk reduction of objective evidence of system performance. Figure 7 illustrates how partial integration helps address that risk, even when full integration is not feasible. This can be accomplished by limiting the scope of integration tests, creating staging environments, and applying virtual or emulated environments, stubs, and mocks.

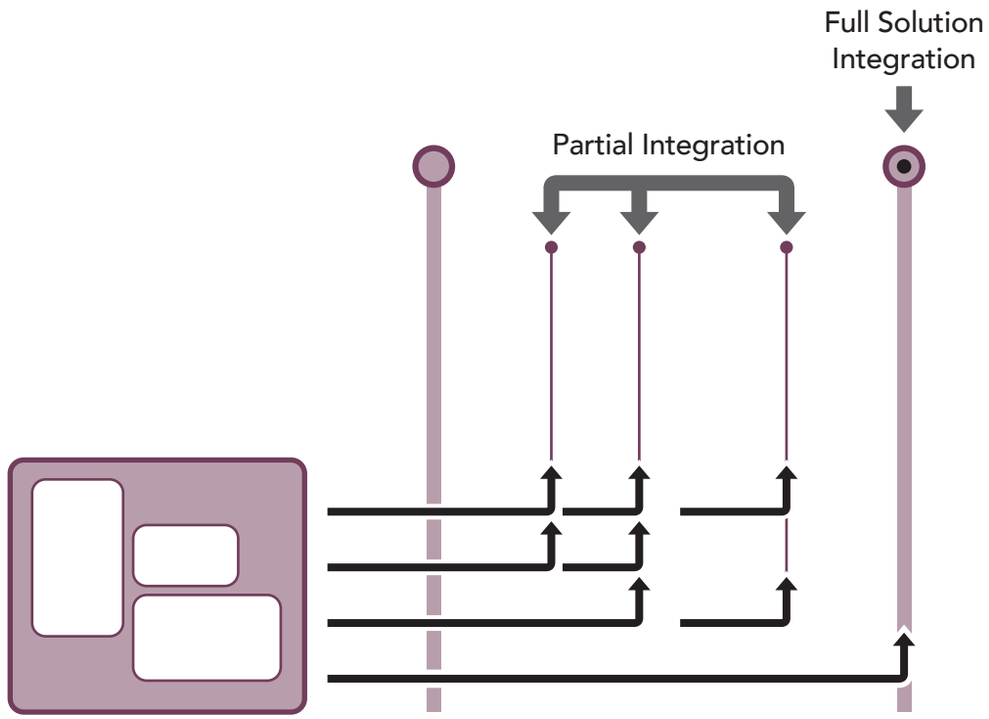

Figure 7: Development Cadence Timebox

In summary, the design of the integration strategy for cyber-physical systems is fundamentally an economic tradeoff, one that weighs the cost of more frequent integration against the risk and ultimate cost of building an inadequate system.

8. Be Test Driven

As described earlier, systems built on modular, component-based architectures that communicate through well-defined interfaces are much simpler to test and verify functional behavior. Each component can be independently built and tested (and in the spirit of DevOps, released) with more confidence that the change does not break another part of the system.

To build component-level tests, engineers apply test-driven development, meaning they write the tests for a change before they implement the change in software or hardware. Writing tests first helps engineers think deeply about the scope of a requirement change before beginning the implementation. Once all tests pass, the work is complete.

Further, these tests should run automatically. In test-driven cultures, the environment automatically runs a rich set of component tests on any change. ECAD and MCAD tools have had testing infrastructure built into them for years. Adopting a test-driven mindset means creating the tests first and running them frequently.

A rich set of component-level tests reduces reliance on larger, slower, system-level tests and dramatically reduces the number of errors caught at the system level. The overall testing goal is to run many automated component level tests frequently on every change and run the larger, slower, more expensive tests less frequently, as shown in Figure 8.

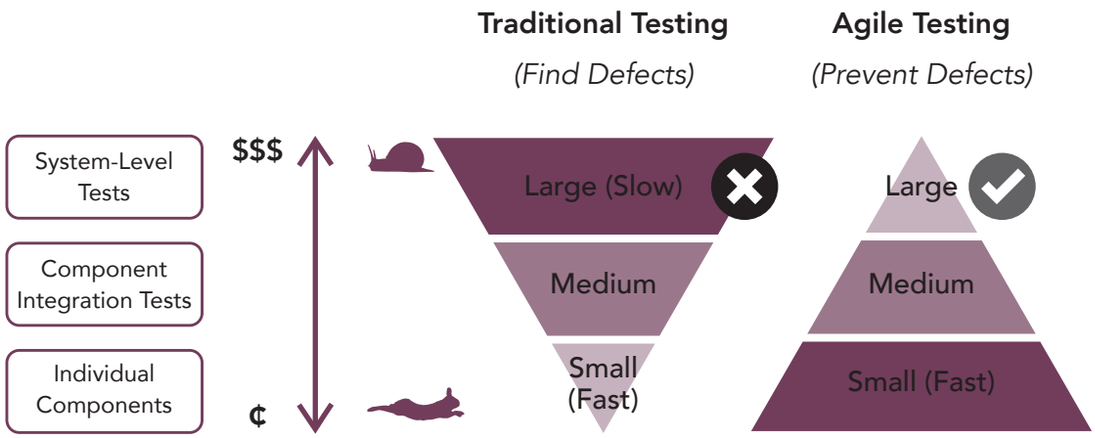

Figure 8: Traditional versus Agile Testing Triangles

DevOps strives for a continuous delivery pipeline, where small engineering changes flow through different levels of testing, potentially all the way to deployment. This is a challenge for hardware, as it is often not available early on, and large systems hardware can be expensive. To address this, teams invest early in building hardware proxies that grow in maturity over time. Proxies include hardware from prior system versions, simulators, development kits, early hardware revisions, digital twins, 3D printing, wood instead of metal, and so on. These proxies allow engineers to validate some assumptions early by continuously integrating and continuously deploying (CI/CD) into the pipeline, as depicted in Figure 9.

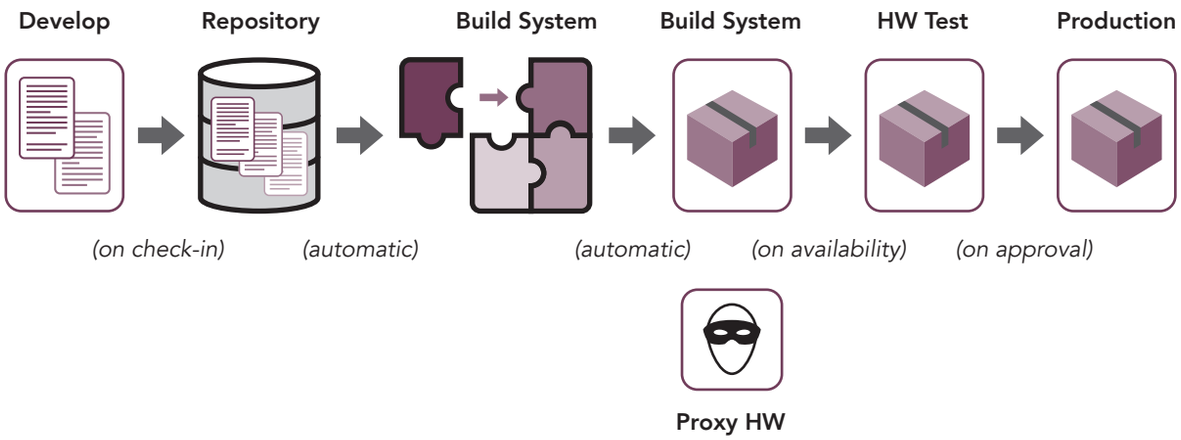

Figure 9: Strive to Add Hardware (Proxies) to the Continuous Delivery Pipeline

Continuous testing environments often require numerous testing platforms. Otherwise, platform availability will become the bottleneck and slow the entire engineering process. It is imperative to understand the economics of investing in proxies and/or sufficient numbers of hardware-testing infrastructure to avoid the cost of delaying value delivery.

CONCLUSION

Applying the theory and practice of learnings from DevOps has the potential to dramatically improve the development of complex cyber-physical systems. Implementing practices such as organizing around value, utilizing multiple planning horizons,

basing system decisions on objective evidence, reducing batch size, architecting for modularity and scale, iterating rapidly for fast feedback, applying cadence and synchronization, "continu-ishly" integrating the entire system, and applying test-driven development methods are keys to succeeding in this endeavor.

The companies that solve this problem first will increase transparency, reduce cycle time, increase value for money, and innovate faster. Simply, they will build better systems faster, and they will become the ultimate economic and value delivery winners in the marketplace.

RESOURCES

Denning, Steve. "What Went Wrong At Boeing?" *Forbes*. January 21, 2013. https://www.forbes.com/sites/stevedenning/2013/01/21/what-went-wrong-at-boeing/#17cb14207b1b.

Goldratt, Eliyahu. *The Goal*. Great Barrington, MA: North River Press, 1984.

Hackster.io. https://www.hackster.io/.

Harnish, Verne. *Scaling Up: How a Few Companies Make It and Why the Rest Don't*. Ashburn, VA: Gazelles Inc., 2014.

Horvath, Kristof. "Using Agile in Hardware Development & Manufacturing?" Intland Software blog. June 15, 2016. https://content.intland.com/blog/agile/using-agile-in-hardware-development-manufacturing.

Manifesto for Agile Software Development. 2001. http://www.agilemanifesto.org/.

McChrystal, Stanley. *Team of Teams: New Rules of Engagement for a Complex World*. London: Portfolio/Penguin, 2015.

"Need for ALM-PLM Integrations." Kovair blog. February 23, 2018. https://www.kovair.com/blog/need-for-alm-and-plm-integrations/.

Oosterwal, Dantar P. *The Lean Machine: How Harley-Davidson Drove Top-Line Growth and Profitability with Revolutionary Lean Product Development*. New York: AMACOM, 2010.

Reinertsen, Donald G. *Principles of Product Development Flow: Second Generation Lean Product Development*. Redondo Beach, CA: Celeritas Publishing, 2009.

"SAFe Principles." Scaled Agile Framework website. October 28, 2017. https://www.scaledagileframework.com/safe-lean-agile-principles/.

Spirent Communications. "Case Study—Implementing DevOps for a Complex Hardware/Software-Based Network Product." SlideShare.net. November 9, 2015. https://www.slideshare.net/Spirent/case-study-implementing-devops-for-a-complex-hardwaresoftwarebased-network-product.

"The Soul of a New Machine." Wikipedia. April 15, 2018. https://en.wikipedia.org/wiki/The_Soul_of_a_New_Machine.

Ulanoff, Lance. "Inside Apple's Perfectionism Machine." Mashable. October 28, 2015. https://mashable.com/2015/10/28/apple-phil-schiller-mac/#lmfqMrPzMaql.

CONTRIBUTORS

- Dr. Suzette Johnson, Fellow and Agile Transformation Lead, Northrop Grumman Corporation
- Diane LaFortune, Design Data and Product Integrity Engineering Manager, Northrop Grumman Corporation
- Dean Leffingwell, Co-founder and Chief Methodologist, Scaled Agile Inc., @deanleffingwell
- Harry Koehnemann, SAFe Fellow and Principal Contributor, Scaled Agile Inc.
- Dr. Stephen Magill, Principal Scientist, Galois, Inc.
- Dr. Steve Mayner, SAFe Fellow and Principal Consultant, Scaled Agile Inc., @stevemayner
- Avigail Ofer, Senior Director Communications and Digital Marketing, Electric Cloud
- Anders Wallgren, CTO, Electric Cloud, @anders_wallgren
- Robert Stroud, CPO, XebiaLabs
- Robin Yeman, Lockheed Martin Fellow, Lockheed Martin Corporation, @robinyeman