Breaking the Change Management Barrier

Patterns to Improve the Business Outcomes of Traditional Change Management Practices

IT REVOLUTION
DEVOPS ENTERPRISE FORUM
2019

25 NW 23rd Pl
Suite 6314
Portland, OR 97210

Breaking the Change Management Barrier

This work is licensed under the Creative Commons Attribution-NonCommercial -ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

When sharing this content, please notify
IT Revolution Press, LLC, 25 NW 23rd Pl, Suite 6314, Portland, OR 97210

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.

The 2019 DevOps Enterprise Forum was sponsored by XebiaLabs.

Preface

In May of this year, the fifth annual DevOps Enterprise Forum was held in Portland, Oregon. As always, industry leaders and experts came together to discuss the issues at the forefront of the DevOps Enterprise community and to put together guidance to help us overcome and move through those obstacles.

This year, the group took a deeper dive into issues we had just begun to unpack in previous years, providing step-by-step guidance on how to implement a move from project to product and how to make DevOps work in large-scale, cyber-physical systems, and even a more detailed look at conducting Dojos in any organization. We also approached cultural and process changes like breaking through old change-management processes and debunking the myth of the full-stack engineer. And of course, we dived into the continuing question around security in automated pipelines.

As always, this year's topics strive to address the issues, concerns, and obstacles that are the most relevant to modern IT organizations across all industries. Afterall, every organization is a digital organization.

This year's Forum papers (along with our archive of papers from years past) are an essential asset for any organization's library, fostering the continual learning that is essential to the success of a DevOps transformation and winning in the marketplace.

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 and the weeks that followed to help everyone stay focused and productive. Additional thanks goes to this year's Forum sponsor, XebiaLabs. And most importantly a huge thank you to this year's Forum participants, who contribute their valuable time and expertise and always go above and beyond to put together these resources for the entire community to share and learn from.

Please read, share, and learn, and you will help guide yourself and your organization to success!

—Gene Kim
June 2019
Portland, Oregon

Collaborators

Ron Forrester, Sr.
Director, Partnership Engineering, Nike

Jayne Groll
CEO, DevOps Institute

Chris Hill
Senior Manager Continuous Delivery Platform, T-Mobile

Dr. Steve Mayner
SAFe Fellow and Principal Consultant, Scaled Agile, Inc.

Erica Morrison
Executive Director Software Engineering, CSG International

Scott Nasello
Director, Delivery Engineering, Columbia Sportswear

Scott Prugh
Chief Architect & SVP Software Engineering, CSG International

Rosalind Radcliffe
Distinguished Engineer, IBM

Introduction

During an era of rising regulation and linear, waterfall software development, enterprises adopted a rigorous approach to IT's change-management process in order to regulate releases into production systems and demonstrate auditable Information Technology (IT) controls. For the purposes of this paper, the term "change management" refers to enterprise technology changes.

Best practice frameworks, such as the Information Technology Infrastructure Library (ITIL), advocated for the creation of Change Advisory Boards (CABs) that would be responsible for assessing requests for change (RFC) against risk and impact, as well as collision avoidance. The intent was to create a holistic advisory group with the skills and experience to evaluate "normal" changes and strike a balance between stability and innovation. By definition, normal changes were unique and had no history of risk or reward.

As change management became entrenched in the enterprise, the CAB shifted from an advisory group to a decision authority for most, if not all, requested changes. The burden on the change initiator grew as RFC details, timelines, and level of pre-submission approvals increased.

Traditional change management places more emphasis on managing RFCs over the change traceability, or change record. The net result has made for longer lead times, increased impediments and overhead costs, and frustration from the Development and Operations teams, business leaders, and customers. Once an enabler of innovation, command-and-control change management is now considered a constraint.

The business climate has changed, and IT must adapt its processes accordingly. IT has to implement changes more rapidly to gain or sustain a competitive advantage in this disruptive landscape. Smaller releases with risks mitigated by frequent and automated testing can

deliver value faster and more frequently if allowed to deploy into production with a minimum viable process. Determining what's "just enough" change management depends on the risk appetite and compliance requirements of the organization. However, even small improvements to change management and the CAB can result in big advancements.

This paper offers several patterns that can be applied in tandem or as appropriate by leaders seeking ideas and opportunities for optimizing the ongoing value of change management while reducing its complexity.

What Business Leaders Care About

Any discussion of transforming the change-management process must begin with a recognition that legacy change practices were put in place for a reason. Boards of directors and officers of corporations have fiduciary accountability for ensuring the ongoing viability of the organization. Public-sector leaders are similarly responsible for the use of taxpayer funds to provide vital services to citizens. These responsibilities frequently involve compliance with a complex spectrum of laws and regulations. As organizations become increasingly dependent on technology for critical business capabilities, system failures represent an unacceptable threat to them and to society, requiring business leaders to institute measures that anticipate and reduce these risks.

More specifically, laws such as the Sarbanes-Oxley Act of 2002 in the United States, international standards bodies such as the Payment Card Industry (PCI) and International Standards Organization (ISO), and regulatory agencies such as the Food and Drug Administration (FDA) prescribe adherence to change management best practices as a compliance requirement. The governing bodies commonly write standards with a bias toward traditional, process-heavy change practices. Executives, compliance officers, and internal and external auditors can all be held personally accountable for failure to comply to these guidelines. This frequently, and understandably, leads to a range of responses, from skepticism to outright rejection, when considering proposals for implementing modern change management practices.

IT leaders including chief information officers (CIOs), chief information security officers (CISOs), program managers, technical managers, and even teams themselves are

invested in ensuring changes are applied to production systems in ways that don't result in outages, invalid data, poor responsiveness, and more. Failures in critical technology capabilities lead to a loss of confidence in the IT organization, as well as the diversion of limited resources away from innovation to recovery and remediation activities.

Dysfunctions with Traditional Change Management

Well-intentioned, traditional change-management processes frequently display a series of dysfunctions and anti-patterns. They include:

1. **Disjointed ownership of workflow and responsibility:** many different teams and individuals are involved, and flow is sacrificed.
2. **Large batches of work:** changes are queued with long lead times.
3. **Unproductive CAB meetings:** with so many changes reviewed at each meeting by several individuals, meetings can become watered down and valuable time that could be invested elsewhere is wasted.
4. **Approval removed from accountability and responsibility:** those farthest from the change have the approval authority; those closest to the change are removed from the approval process.
5. **Resource dedication to "process excellence" and job security:** incentives to keep things as they are remain strong.
6. **Tool silos or tool swivel:** change management tools often differ from team level work management tools.
7. **Overuse of change moratoriums and freezes:** leaders believe that not changing the system results in stability; in reality, batching up work introduces risk, as does forcing teams to defer or reorder change implementations.
8. **Lack of continuous improvement capabilities:** while continuous improvement has permeated many other aspects of DevOps organizations, it's not routinely applied to change management.
9. **Multiple approval theater:** those farthest from the issues are allowed to weigh in and approve or deny changes; there is a sense that by the simple act of getting to approve changes, safety will be introduced.
10. Decisions are often made based on emotion instead of metrics.

Overarching Patterns to Improvement

Before exploring the specific patterns in evolving the change-management process for modern systems development, it's important to address the foundations that provide a principled backdrop for each of the below recommendations.

Embrace Lean Thinking

The first overarching pattern is to understand and embrace Lean thinking. "Lean" refers to a body of knowledge that was first pioneered in manufacturing by Toyota in the 1940s. It describes a set of principles, practices, and tools that enable people to continually improve their work.[1] These improvements focus on reducing waste and increasing quality.

The concepts of Lean have been adapted to technology development by Allen Ward,[2] Don Reinertsen,[3] and others. In addition to reducing waste and increasing quality, the application of Lean to product development aims to create a continuous flow of value in the form of working systems and software features. While these thought leaders have described many concepts and practices for Lean product development, three are particularly relevant to the discussion of rethinking the traditional change-management process.

Large Batches Create Risk

One of the keys to creating continuous flow of value in product development is to reduce the batch size of work. Research cited by thought leaders, such as Reinertsen, shows that smaller batches move through the end-to-end development process quicker, with less variability and fewer defects. In the context of change management, smaller, more frequent changes represent less risk as there are fewer variables that have the potential to result in an error or outage. Smaller batches are easier to test, and defects are easier to find and correct.

Queues Increase Lead Time

Another important concept from Lean is that longer queues increase the lead time in which it takes new features to flow through the process from inception to delivery.

This principle is described in the queuing theory known as Little's Law that states the average wait time for a service from a system equals the ratio of the average queue length divided by the average processing rate. In other words, the longer the queue, or list of committed items in the backlog, the longer the wait time for customers to receive the value represented by the new feature.

Applied to change management, long queues of changes cause delays. Therefore, the change-management process needs to be optimized to commit to short lists of changes, while all other changes are simply "forecasted." This maximizes the flexibility to add, delete, or re-sequence changes to deliver the highest value changes in the shortest lead time. Not only do queues create long lead times, but they also create an opportunity for lost context and knowledge as the work is handed off from one team to another. We like to say that: "Not only do queues not scale, but they also don't learn."

Experiment and Learn

An impediment to proposing a new approach for change management is the daunting prospect of replacing deeply rooted processes that were once widely accepted as best practice. It's ironic that the fear of change is one of the biggest barriers to improving *how* we change. Fortunately, the principles of Lean and Agile once again provide a path forward. Each of these bodies of knowledge draw from the language of the scientific method by initiating change as a hypothesis, followed by experiments testing that hypothesis. Once the experiments have concluded, observations and learnings develop the next hypothesis based on the results. The guidance here is to evolve change-management practices as a series of small experiments, enabling the organization to iteratively understand and adapt these processes over time based on validated learning and measured improvements in quality and flow.

Empower the Teams

The next foundational pattern's purpose is to empower teams to participate in and contribute to the development of modern change-management practices. After all, the teams generate changes that ultimately flow into the production system and have valuable insights into techniques such as automation that can manually provide

previously performed validations. Not only do they provide better solutions for improving change management but being a part of the improvement process also engenders ownership and accountability in the teams for the overall execution. Lyssa Adkins, a well-known thought leader in Agile coaching, states, "Accountability sticks when you mindfully work towards getting to a solution rather than passively receiving directions from another."[4]

Eat the Edges Incrementally

We understand that many organizations have deeply entrenched change processes that have evolved over years and even decades. This includes lengthy CAB meetings, multiple approvers, and dedicated people who manage these changes. Disrupting this head-on will be difficult. With this paper, our hope is to provide a set of patterns that can be applied incrementally to improve change in your organization. We recommend an approach that "eats the edges incrementally" of your change-management process. For example: start with applying easier patterns, such as *low-risk classing and exemptions* to reduce overhead and gain wins. Then, begin moving to more complex patterns, such as *localizing changes and CAB*. By this manner, you can deliver both value and credibility faster. Attacking the entire change process at once will likely lead to setbacks, and we do not recommend starting with that approach.

Patterns

Figure 1 represents a set of patterns that can be adopted in support of the transformation. Each pattern provides a brief summary of the pattern, the problem it's addressing, and a proposed solution. Our goal is to demonstrate ways organizations have transformed and to help provide specific guidance that will improve the change process in your organization. Where applicable, we also include concrete examples of what has worked in other organizations. The patterns are introduced by increasing difficulty of adoption for traditional organizations with entrenched change processes.

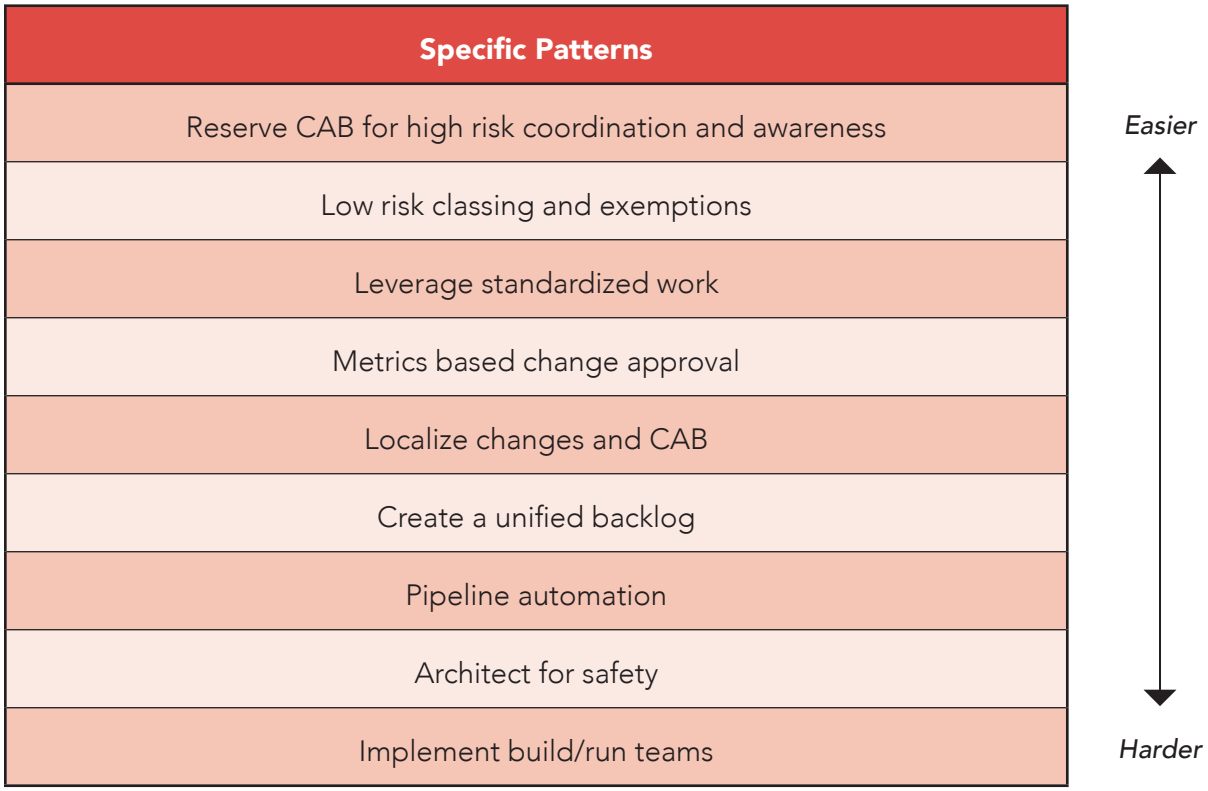

Figure 1: Change Management Improvement Patterns

Reserve CAB for High-Risk Coordination and Awareness

1. Summary

Limit CAB to review only high-risk, high-impact changes and allow lower-risk changes to flow more quickly. This allows higher-risk changes to gain better visibility and optimizes CAB usage and engagement.

2. Problem

In many enterprises, the Change Advisory Board, sometimes referred to as the Change Approval Board, is often used to evaluate risk and awareness for all changes, including those that can be handled better by individual teams. This creates large batches of work and recurring meetings. It also results in low-risk change flow reduction. Additionally, high-risk changes don't get the focus they should because they're caught up in the noise of a large batch of changes.

While this gives the illusion of safety, it's actually counterproductive to improving safety and risk assessment. Time for many of the leaders involved could be better utilized by partnering directly with their teams. Those with the most knowledge about the changes of implementing teams have the least say in the process.

3. Intent and Business Goals

- Provide impact and risk assessment for enterprise-wide and high-risk changes.
- Provide discussion forum and awareness for high-risk changes.
- Provide mitigation opportunities for the riskiest changes.
- Create flow for lower-risk changes to be handled elsewhere.
- Optimize time and productivity to focus on riskiest changes.
- Identify optimal implementation timing and collision avoidance.

4. Solution

- Shrink the timebox and frequency of CAB.
- Shift the tone of CAB back to advising and awareness, not approval.
- Only review high-risk, high-impact changes in CAB.
- Publish agenda ahead of time, so key participants are present.
- Put responsibility on the change implementer to pull through key participants.
- Put responsibility on the change implementer to build awareness prior to CAB.
- Only meet when necessary and cancel CAB as appropriate.
- Develop a lightweight, out-of-cycle, and typically offline CAB process for items that absolutely cannot wait until the next scheduled CAB.
- For lower-risk changes that bypassed CAB and resulted in outages, perform a retrospective and share key observations with CAB members.

5. Other Patterns to Consider

- Low-risk classing and exemptions
- Leverage standardized work
- Localize changes and CAB

6. Examples

- Shrink the Timebox and Frequency of CAB: CAB is shrunk to a thirty-minute timebox once a week and only high-risk, large surface area changes are discussed

for awareness. Scheduling conflicts with changes are resolved in this meeting. An emergency CAB is held as needed to discuss high-risk, large surface area changes that can't wait until the next weekly meeting.

A daily fifteen-minute morning stand-up is held with all Operations and Development personnel to review all incidents and critical changes coming up. This creates a flow of awareness for high-risk changes and minimizes the need for CAB. As such, the weekly CAB meeting can often be cancelled.

Low-Risk Classing and Exemptions

1. Summary

Leverage risk classification to exempt low-risk changes from review and approval processes and incentivize teams to promote standard work and automation for exemption.

2. Problem

In many organizations, change-management activities tend to gravitate toward a one-size-fits-all approach, where minor changes are treated the same as major changes. Treating all changes the same becomes the standard operating procedure, as any exceptions may potentially lead to a slippery slope where more and more changes are not following the prescribed IT Governance Controls framework.

Unfortunately, the one-size-fits-all approach creates an incentive for technical teams to undermine the spirit of a robust change-management practice as they may go out of their way to not follow the process. As more changes occur that don't follow the sanctioned practice, the change-management leadership may respond in a disproportionately strict manner to ensure process compliance.

3. Intent and Business Goals

- Implement appropriate processes and controls to minimize potential for business disruption.
- Create focus on high-risk changes and collisions by moving low-risk changes through the system faster with little to no ceremony.

4. Solution

- Optimize processes for low-risk change throughout. Rather than define processes and activities for the worst-case scenario, major risk and major impact, look for ways to effectively and efficiently handle lower-risk changes outside of CAB and with lighter-weight RFC processes. This can include:
  - Leverage risk classification to correctly identify the change.
  - Exempt lower-risk changes from multiple approver chains.
- Prefer intra-team approval/peer review or leverage readily available partners, such as having a developer approve an operations change and vice versa.
- Look to increase risk classification for change types or teams who have recurring failures.
- Teams with a history of unexpected impact to other teams may lose eligibility to approve and execute changes locally.
- Leverage standardized work to drive lower-risk classing. The ITIL recognizes multiple types of changes: normal, standard, emergency, and informational. (See the section "Leverage Standardized Work" for more details.)
- Leverage business exempt changes. Define a catalog of changes that can be done outside the change-management process. Examples include configuration, such as pricing, and descriptions or images that are often executed through a controlled user interface (UI). These changes should be self-service from a catalog and executed by business partners. If possible, make these changes easily visible by logging the change in the tool or chat room. Ideally, make it easy to roll the change back in case there is an unintended side effect.

5. Other Patterns to Consider

- Reserve CAB for high-risk coordination and awareness
- Leverage standardized work

Leverage Standardized Work

1. Summary

One of the most powerful concepts of Lean is standardized work. The goal of standardized work is to provide a basis for understanding, measurement, and improve-

ment. By improving the level of standardized work, we can create an environment of continual learning and improve operational quality. We can then use this to lower the burden of the change process.

2. Problem

In several enterprises, the operational tempo of the day-to-day prevents teams from evaluating existing activities by the merits of the opportunity cost, purpose, and efficacy of the work. Consequently, legacy work is never classified, refactored, or streamlined for better business outcomes. Organizations don't leverage the concept of standardized work to lower the burden of the change process.

3. Intent and Business Goals

The most important aspects and business goals of standardized work in the change-management process are to provide:

- A basis to document current best practices.
- A basis to measure performance, such as lead time, failure rate, and MTTR (mean time to repair).
- A basis to communicate business intent of the change.
- A construct to justify low-risk classing and exemptions.
- Most importantly, a basis to continually improve upon and develop new best practices.

4. Solution

- Identify the type and value of the work.
- Map the value stream of the work.
- Recognize and define the work standard.
- Leverage work templates, if available.
- Measure lead time of work type in the system.
- Track failure rate and MTTR of work type failures.
- Seek low-risk classing for work types that are templated and show improvement.
- Iterate on improving both lead time and MTTR for work type.
- Reduce friction for standard changes. Standard changes are envisioned as having lightweight processes. In many enterprises, enrolling changes as standard

may have become more difficult than necessary. Create a key performance indicator (KPI) for the percentage of changes following the standard change path. Explore the following opportunities for simplification:

  - Discontinue the use of expiration dates for standard changes and revisit previously expired standard changes for reinstatement.
  - Reduce the documentation requirements for standard changes.
  - Create a blameless postmortem process for failing standard changes so that the efficacy of these changes can be improved and don't lose standard change eligibility.
  - Coach technology teams on how they can decompose their changes to be candidates for standard change.
  - Assess whether the most consistent normal changes are candidates for standard change.
  - Streamline ITSM steps and flow. Logging standard changes in the ITSM system should be possible in less than a minute and with only a click or two. The API may be able to further simplify the practice.
- Add system robustness to remove the need for standardized work. Look for ways to change and improve the system so that specific types of work no longer need to be done. Not having to do the work in the first place is much better than managing standardized work.

5. Other Patterns to Consider

- Low-risk classing and exemptions
- Create a unified backlog
- Localize changes and CAB
- Implement build/run teams

6. Examples

- Recognize and define the work standard: A build/run team with a shared backlog notices a bimonthly password change that is executed as a high risk and requires multiple approvers. The team begins by first documenting the best practice to execute the change and then defines a "password change" template

in the ITSM tool so that this change can be classified as standardized work and moved to low risk. Once it's standardized and steps are understood, the team automates the change to improve lead time and reduce risk. This change is no longer discussed in CAB and no longer requires multiple approvers.

- Add system robustness to remove the need for standardized work: A build/run team notices that when back-end middleware servers are patched and rebooted, they have to coordinate a dependent server reboot to reconnect to those servers. The team first classifies that work as a standard "dependent server reboot," then they re-engineer the connectivity to auto-connect to the server once it is rebooted. This way, an entire class of change no longer needs to be managed.

Metrics-Based Change Approval

1. Summary

Use the right trended metrics to inform your CAB/change approval process and lower change burden for teams who demonstrate success with these metrics.

2. Problem

With the increased pace of change in today's enterprises, it's no longer viable, if it ever was, to have humans manually extract, observe, and assess the state of our software and infrastructure. The lack of consistent, trended, and automatically generated key performance metrics for our systems leaves organizations blind to the risks involved with rapidly deploying changes into their environments. Organizations that have a CAB, or equivalent, may not be utilizing trended metrics to inform decisions that approve or deny changes to software or infrastructure. This can be due to the lack of visibility of existing trended metrics, or the lack of those metrics altogether.

3. Intent and Business Goals

- Make metrics a visible part of the change approval process in order to quantify risk.
- Using the metrics, where possible, move from a human gated change process, or CAB, to an automated pipeline change process.

4. Solution

- Identify key metrics to track and trend:[5]
  - Lead time for change.
  - Standard work percent.
  - Templated work percent.
  - Change failure rate.
  - MTTD change failure.
  - MTTR from change failure.
  - Change failure rate by team.
  - Change failure rate by work type: standard vs. normal, templated work.
- Develop a consistent source of truth for each metric.
- Develop the processes and tools to generate historical trends for each of the metrics at the right organizational granularity.
- Make these metrics available via both an API for automated change approval, and visualized through a dashboard and/or reports for human based change approval.
- Inject these metrics into the change-decision process, whether automated or CAB based. Examples include:
  - Track and improve standard work percentage: Categorize change using the definition of standardized work and track the percentage of work that falls into that category with the goal of making as much work as possible standard. Standardized work can then be used as a means to bypass the CAB process, and the CAB can focus on unstandardized work and the additional risk it implies.
  - Improve MTTD and MTTR from change failures: Change failure is an inescapable reality when dealing with change in complex systems. While it's important to work to reduce change failure, it's arguably more beneficial to focus on reducing the mean time to detect (MTTD) a failure and reducing the time to recover. Tracking these metrics helps to uncover gaps in monitoring and alerting, troubleshooting, and repeatable automated deployment processes and related tools. When a team has accurately tracked and trended their MTTD and MTTR, it's possible to predict the risk of changes deployed by that team.

5. Other Patterns to Consider

- Pipeline automation
- Architect for safety
- Localize changes and CAB
- Low-risk classing and exemptions

Localize Changes and CAB

1. Summary

By holding local CABs, change control authority and responsibility is placed into the hands of those closest to the problem and with the most knowledge to affect safety.

2. Problem

Individuals, such as operational engineers, and teams who are responsible for implementing and validating change are not traditionally involved in the CAB process. That means those with the most firsthand knowledge of the change, impacts, and preparedness are left out of the discussion. Not only does this result in an incomplete picture presented at CAB, but it can also mean members of the same team are unaware of all the changes that are going on.

3. Intent and Business Goals

- Provide a mechanism to decentralize change approval and coordination.
- Provide a mechanism to place change approval closest to responsibility and accountability.
- Support enterprise requirements of overall visibility, auditability, and risk management.

4. Solution

- Implement local CABs at the individual team level.
- Hold local CABs at a frequency that makes sense for each team. For high frequency change teams, local CABs meet daily. This can often be a daily stand-up combined with other reviews or stand-ups.
- Schedule local CABs within one to two days of going live for the change. Introducing more lead time has been shown to result in a loss of knowledge.

- Invite all team members to the local CAB.
- For changes that impact and/or require validation from other teams, invite them to the local CAB as well.
- Discuss the following key components:
  - Details of the change.
  - Validation that peer review was completed.
  - How the change was tested.
  - How the change will be validated.
  - Rollback plan.
- Keep documentation level to only what is necessary. The focus is on the conversation. For historical purposes, lightweight notes may be helpful to capture the conversations.
- Consider tooling most familiar to DevOps teams. This may mean something similar to the JIRA software instead of formal change-management tools. Formal change-management tools are still necessary, but local CABs often evolve better when keeping tooling lightweight for the actual discussion.
- Grant local CAB authority to deny or approve changes. Default to rescheduling changes if issues are discovered during CAB review.

5. Other Patterns to Consider

- Reserve CAB for high-risk coordination and awareness
- Low-risk classing and exemptions
- Leverage standardized work
- Create a unified backlog
- Architect for safety

6. Examples

- Implement local CABs at the individual team level: Hold a daily change stand-up for the team, or teams, that own the service. This can be a joint stand-up with the feature backlog or a separate stand-up to just discuss changes. The entire team reviewing should consist of DevOps engineers to create awareness for the operational complexities of running the service. Additionally, the team should use tools they are most comfortable with, such as JIRA or physical boards, to

discuss the changes. The team will walk through each change discussing: (1) change details, (2) peer review, (3) change testing in lower environments, (4) change validation upon deployment, and (5) rollback plan upon change failure.

Create a Unified Backlog

1. Summary

A unified backlog allows for visibility and prioritization of the full body of work required to operate a service and incentivizes improvements for repetitive and risky work.

2. Problem

Oftentimes, each type of work, including features, changes, service requests, incidents, and security, are all managed in different systems as different backlogs and by different teams. This results in:

- Siloed optimizations where workers on each backlog are kept busy.
- Irregular flow of the work where context is lost in handoffs and queues.
- Long lead times for change as tickets are handed off from one team to another and one system to another.
- Failure to allocate time to systems and change improvements because developers are focused on features.
- Fast fixes and closure of incidents that fail to address causal factors and improve the system due to incentives such as "first call resolution" and the minimization of "call handle time."
- The end result is operational instability and brittleness, especially around the area of change.

3. Intent and Business Goals

- Provide a mechanism for stakeholders to understand the body of all work that is required to enhance and maintain a service.
- Provide a mechanism for stakeholders to prioritize across the value stream of work as opposed to silos of work.
- Provide a mechanism for teams to not just execute the work but improve the work that is required to operate a service.

4. Solution

- Make work visible by unifying all items into one backlog via consolidation or synchronization.
- Manage investment priorities across the unified backlog, not just features.
- Optimize repetitive and risky work that is visible in the backlog to make the system safer to operate and change.

5. Other Patterns to Consider

- Architect for safety
- Implement build/run teams

6. Examples

- Make work visible by unifying all items into one backlog via consolidation or synchronization. (See the 2018 DevOps Enterprise Forum paper *Overcoming Inefficiencies in Multiple Work Management Systems* for more details.[6])
- Manage investment priorities across the unified backlog, not just features: When managing the entire backlog, pay special attention to operational activities that are recurring or could be considered toil. Work with your business stakeholders to carve off a specific capacity to address these concerns. This can also be a fixed capacity reserve for each release, so that it's always available for improvement. At CSG International, 15% of total capacity is dedicated to work–life balance efforts and is reserved for the teams to improve their service.[7]
- Collapse separate DevOps teams, as well as backlogs (see Figure 2).

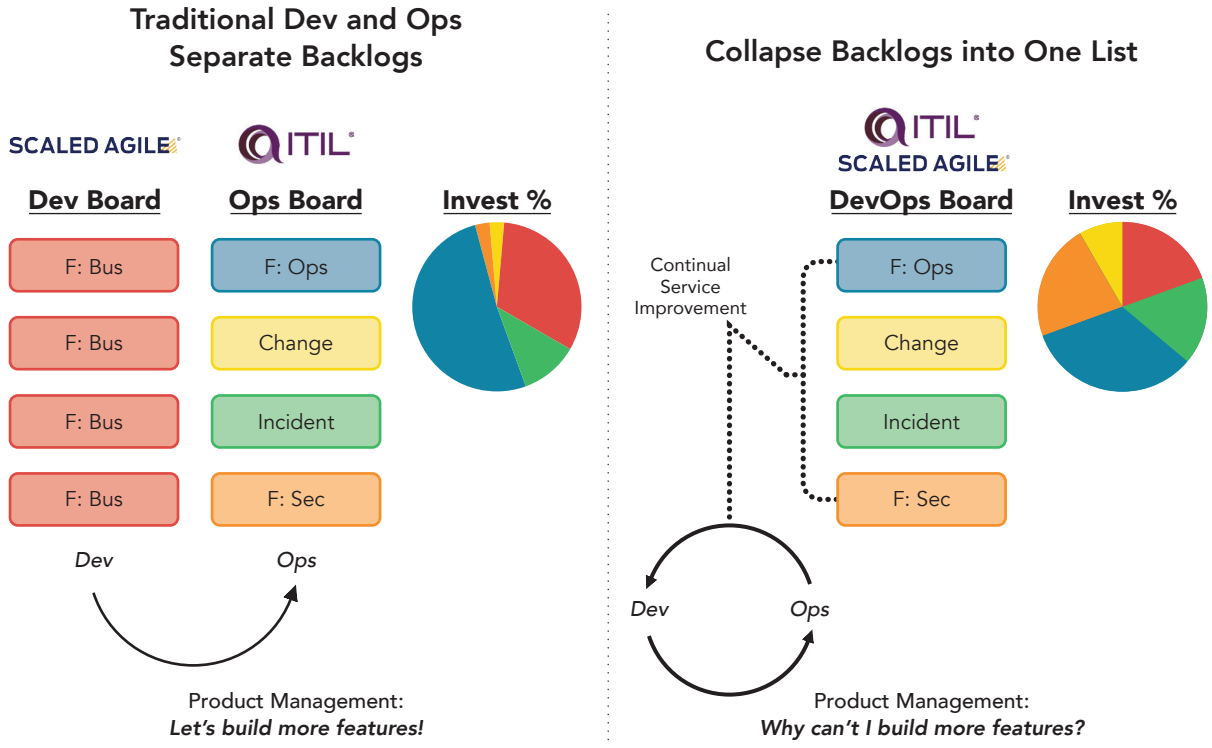

Figure 2: Team owns features, service requests, incidents, change and security. Integrate operations concerns into construction instead of after deployment.

Pipeline Automation

1. Summary

By taking the "shift left" approach to the change process and applying pipeline automation, we can greatly decrease the burden of change management and lower risk in change execution.

2. Problem

The goal of the change process is to manage and reduce the underlying risk of a change, as well as bring visibility to those changes. Confidence in this process comes in the form of accurate descriptions, test results, problem statements, and planned deploy schedules. Often the act of documenting the change is burdensome and inaccurate due to the swivel effect of pulling information from SDLC tools and repositories.

3. Intent and Business Goals

- Reduce the overhead of managing the change process for teams.
- Leverage inherent details about the change from other systems, such as code repositories and build pipelines.
- Increase visibility and accuracy for changes flowing through the systems.
- Shift change responsibility and execution to the teams.
- Establish self-service capabilities for the change initiators.

4. Solution

- Implement an API that exposes a change service, so that change requests (CRQs) can be created from software pipelines and lineage tracked directly to changes.
- Treat operational changes as code that's checked into version control and managed in the pipeline.
- Leverage build dependencies to understand risks and dependencies between services.
- Improve change log fidelity by integrating CRQ creation into asset publication steps. For example, when binaries or Docker images are tested and ready for deploy, create the CRQ to stage the asset to production including information such as the location, version, and checksum.
- Integrate change validation into the pipeline by running tests against the newly deployed asset.
- Integrate change rollback into the pipeline by reverting the change if tests fail.
- Integrate CRQ closure into asset publication steps. For example, when the asset is deployed and tests pass, then close the CRQ and include the automated test results in the CRQ notes.
- Consider integrating the change service to enterprise chat channels or a running log of all changes for greater visibility.

5. Other Patterns to Consider

- Architect for safety

Architect for Safety

1. Summary

We can incorporate the design and need for changes into both our software and operational tooling. This can greatly improve system robustness and lower the risk of change. Operations is an engineering problem.

2. Problem

Many applications have inherently brittle operational characteristics. This makes changes high risk with large blast radiuses. This problem stems from the traditional mindset that operational concerns are an exercise to figure out and are not designed into the software and processes that support it. Additionally, due to the siloed nature of organizations and roles, changes tend to involve many teams in order to implement, creating high coordination overhead.

3. Intent and Business Goals

- Make systems safer for change and lower operational risk.
- Shift operational concerns left in the construction process.
- Insulate consumers from potential impact.
- Establish robustness in the wider ecosystem.
- Decouple team dependencies and lower coordination overhead.

4. Solutions and Subpatterns

- Leverage green/blue or canary for deployments.
- Leverage feature flags to decouple the deploy from activation.
- Shift configuration management left by treating configuration and operational changes like code.
- Develop immutable application deployments and repave or rollback with each deployment.
- Implement self-service operations: Change processes often involve dependencies across teams. This requires difficult scheduling where many teams need to be tagged on the CRQ and have tasks created for them. By creating a self-service for key operational activities, we decouple the team dependencies and improve the flow of change. Some common self-service operations patterns are:

    - Self-service reboots
    - Self-service patching
    - Self-service validation
- Improve infrastructure and application recovery time: Making infrastructure and applications fast at recovery is a good, broad-based mechanism to improve overall MTTR from a variety of failure scenarios. Make reboots and restarts part of the performance testing process and tune restarts to minimize recovery time.
- Require version compatibility, backward and forward, as well as versioned APIs.
- Automate validation on deploy or continuous validation.
- Automate rollback upon failure.

5. Other Patterns to Consider

- Pipeline Automation
- Localize Changes and CAB
- Implement Build/Run Teams
- Leverage Standardized Work

6. Examples

- Leverage green/blue or canary for deployments.
- Use load balancers to direct traffic away from servers during deployment: A common technique we've seen is to have a pool of identical servers behind a load balancer. When deploying, mark a subset of servers down. An easy way to do this is to have the load balancer monitor an endpoint for server status: /server/status.html and return *up* or *down* indicating if traffic should be sent. Upon the deploy, change a portion of the servers to *down*, deploy new changes, test, and validate locally. Then, begin marking servers up by changing the page to return *up*. Validate traffic to a single server through the load balancer, and then activate all other servers. Do the rest of the servers later to catch any potential anomalies. Rollback is easily done by marking the changed servers down.
- Require version compatibility, backward and forward, as well as versioned APIs:
    - When deploying database changes, make your code tolerate of forward versions so the database can be changed independently of the application layer.

    - When modifying an API, deploy a new version and keep current consumers pointed at the old version. Consider using this in tandem with a feature switch to route consumers to the new API when ready.
- Implement self-service operations.
    - Self-service patching and reboots: Oftentimes, we see teams requiring SAs to patch or reboot servers during deployments. This requires high coordination overhead for a repeatable and low-value activity. Have application teams coordinate with the infrastructure teams to provide patching and reboots as a service. By doing this, CRQs can be scheduled easier with no team dependencies.

Implement Build/Run Teams

1. Summary

Unifying the team across the many roles in a workstream facilitates the ability to deliver business value faster, as well as improve operational quality.

2. Problem

The structure of current organizations has separate teams for building, testing, deploying, and running the system with distinct boundaries. This separates the knowledge of change from those actually doing the change (see Figure 3). These groups have their own responsibilities and are measured based on different aspects. The organizational silos develop skills and optimizations to those groups rather than optimizing to the flow of the solution. This siloed organization requires queues between teams which cause wait states and delays, as well as loss of information and knowledge about the actual functioning of the solution. The separate groups have different measurements and competing goals, which leads to additional requests for review before specific stages, such as deployment. Both the deployment team and the team running the system want to control what goes into production to limit the possibility of problems. The CAB provided this visibility, but at a late state, and still limits the understanding of the overall solution. No one really knows the entire application due to the separation of the teams.

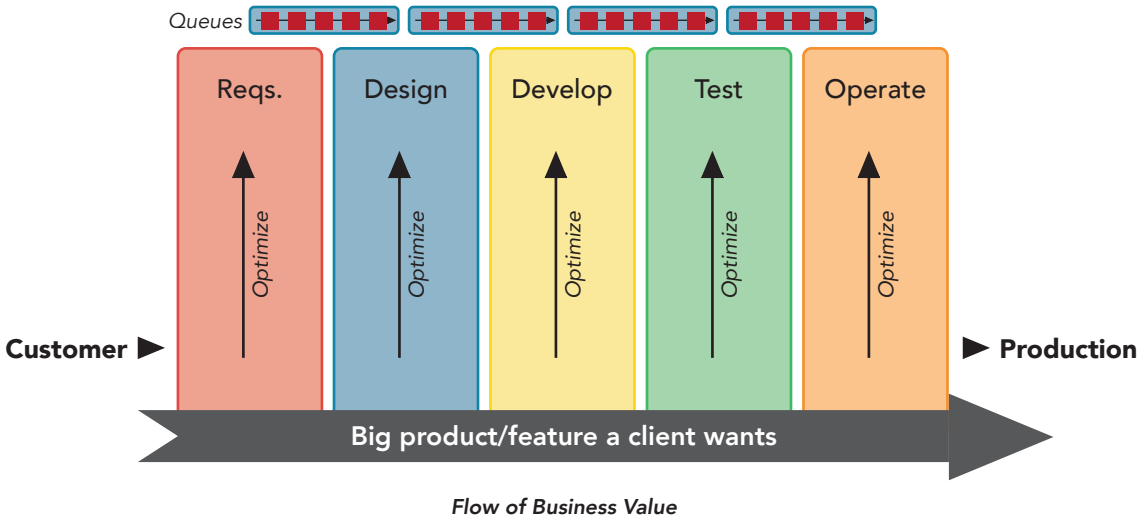

Figure 3: Functional Archetype: Role specific organizations "locally optimize"around their structure and role instead of globally optimizing the flow of business value.

3. Intent and Business Goals

- Faster delivery of business value.
- Removal of wait and handoffs.
- Continuous improvement of operational quality.

4. Solution

- Assign a service owner who has overall responsibility for delivery of business value and operational quality to the solution (see Figure 4).
- Bring the team together with the service owner to have all individuals responsible for building, testing, running, and maintaining the system, with the goal as delivery of business value.
- Provide authority and responsibility to the team for the solution.
- Provide the support and capacity to allow the teams to include continuous improvement of the process and solution within their scope.
- Provide the ability for the team to deploy without external involvement for standard change.

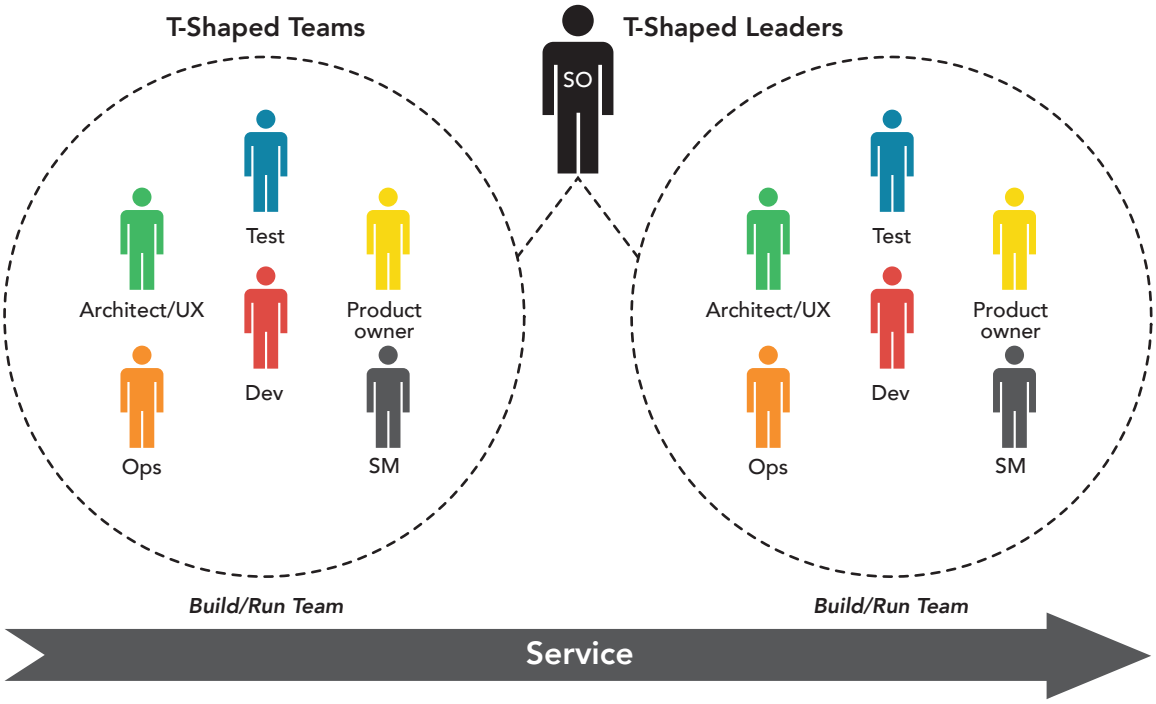

Figure 4: T-Shaped Organization

5. Other Patterns to Consider

- Low-risk classing and exemptions
- Leverage standardized work
- Pipeline for automated change
- Single, unified backlog
- Investment visibility

Conclusion

Traditional change management processes have resulted in longer lead times, increased impediments and overhead costs, and frustration from Dev and Ops, business leaders, and customers. The business climate has changed and the command-and-control change management process no longer drives innovation; it hinders it.

By using the nine patterns detailed in this paper, enterprises can transform their traditional change management practices to embrace modern demands of speed and

stability delivered with DevOps, releasing value over time and setting your enterprise up for success.

References

"Guide to Car Safety Features." Consumer Reports website. Last modified June 2016.

https://www.consumerreports.org/cro/2012/04/guide-to-safety-features/index.htm

Kim, Gene, Jez Humble, Patrick Debois, and John Willis. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR: IT Revolution, 2016.

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

Plumb, Steve. "Steering Towards Autonomy." *Automotive Design & Production*, December 2, 2016. https://www.adandp.media/columns/steering-toward-autonomy.

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, accessed April 29, 2018. https://www.scaledagileframework.com/safe-lean-agile-principles/.

Udaniz, Alex Udanis. "The Technology Behind Active Safety Systems in Cars." *All About Cars*, 2016. https://www.allaboutcircuits.com/news/the-technology-behind-active safety-systems-in-cars/

Notes

1. Jeffrey K. Liker, *Toyota Way: 14 Management Principles from the World's Greatest Manufacturer* (NY: McGraw-Hill, 2003).
2. Allen C. Ward and Durward K. Sobeck, *Lean Product and Process Development,* 2nd ed. (Boston, MA: Lean Enterprise Institute, 2014).
3. Donald G. Reinertsen, *The Principles of Product Development Flow* (Redondo Beach, CA: Celeritas, 2009).
4. Lyssa Adkins, *Coaching Agile Teams* (Upper Saddle River, NJ: Addison-Wesley, 2010).
5. Nicole Forsgren, PhD, Jez Humble, and Gene Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (Portland, OR: IT Revolution, 2018).
6. Scott Prugh, Dominica DeGrandis, Rosaline Radcliffe, Pat Birkeland, and Keanen Wold, *Overcoming Inefficiencies in Multiple Work Management Systems: Helping Your Enterprise with Their DevOps Transformation* (Portland, OR: IT Revolution, 2018), https://itrevolution.com/book/overcoming-inefficiencies-multiple-work-management-systems/.
7. Scott Prugh and Brian Clark, "Product Management Meets DevOps – CSG International," YouTube video, 30:28, posted by IT Revolution, October 24, 2018. https://youtu.be/tgf_D2DUlJ0.

A Special Thank You to Our Sponsor

Our mission for the Forum is to bring together technology leaders across many industries and facilitate a dialogue that solves problems and overcomes obstacles in the DevOps movement. For three days at this private event, we gather 50 of the best thinkers and doers in the DevOps space to tackle the community's toughest challenges. We ask these thought leaders to collaborate and generate a piece of guidance with their best solutions to the challenges.

We would like to thank all of our attendees and our friends at XebiaLabs for helping to make this year's Forum a huge success.