

# MANAGEMENT PRACTICES FOR SOFTWARE

---

**T**he theory and practice of management in the context of software delivery has gone through significant change over the decades, with multiple paradigms in play. For many years, the project and program management paradigm, found in frameworks such as the Project Management Institute and PRINCE2, dominated. Following the release of the Agile Manifesto in 2001, Agile methods rapidly gained traction.

Meanwhile, ideas from the Lean movement in manufacturing began to be applied to software. This movement derives from Toyota's approach to manufacturing, originally designed to solve the problem of creating a wide variety of different types of cars for the relatively small Japanese market. Toyota's commitment to relentless improvement enabled the company to build cars faster, cheaper, and with higher quality than the competition. Companies such as Toyota and Honda cut deeply into the US auto manufacturing industry, which survived only by adopting their ideas and methods. The Lean philosophy was initially adapted for software development by Mary and Tom Poppendieck in their

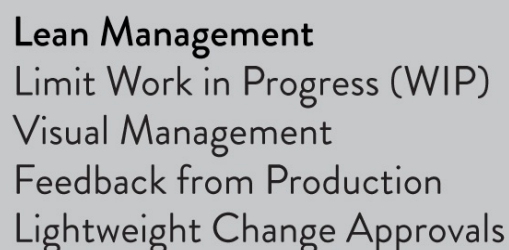
*Lean Software Development* book series.

In this chapter, we discuss management practices derived from the Lean movement and how they drive software delivery performance.

## LEAN MANAGEMENT PRACTICES

In our research, we modeled Lean management and its application to software delivery with three components (Figure 7.1 along with lightweight change management, discussed later in this chapter):

1. Limiting work in progress (WIP), and using these limits to drive process improvement and increase throughput
2. Creating and maintaining visual displays showing key quality and productivity metrics and the current status of work (including defects), making these visual displays available to both engineers and leaders, and aligning these metrics with operational goals
3. Using data from application performance and infrastructure monitoring tools to make business decisions on a daily basis



**Lean Management**  
Limit Work in Progress (WIP)  
Visual Management  
Feedback from Production  
Lightweight Change Approvals

*Figure 7.1: Components of Lean Management*

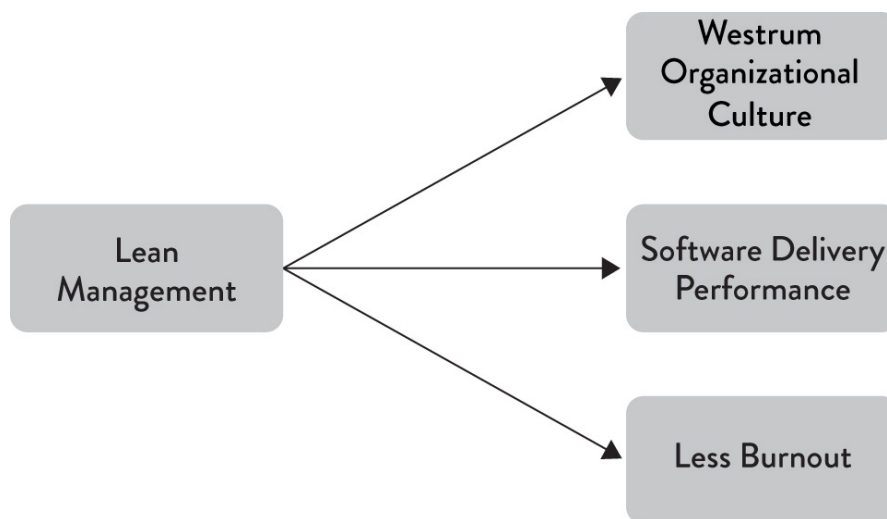
The use of WIP limits and visual displays is well known in the Lean community. They are used to ensure that teams don't become overburdened (which may lead to longer lead times) and to expose obstacles to flow. What is most interesting is that WIP limits on their own do not strongly predict delivery performance. It's only when they're combined with the use of visual displays and have a feedback loop from production monitoring tools back to delivery teams or the business that we see a strong effect. When teams use these tools together, we see a much stronger positive effect on software delivery performance.

It is also worth going into a bit more detail on what exactly we're measuring. In the case of WIP, we're not just asking teams whether they are good at limiting their WIP and have processes in place to do so. We're also asking if their WIP limits make obstacles to higher flow visible, and if teams remove these obstacles through process improvement, leading to improved throughput. WIP limits are no good if they don't lead to improvements that increase flow.

In the case of visual displays, we ask if visual displays or dashboards are used to share information, and if teams use tools such as kanban or storyboards to organize their work. We also ask whether information on quality and productivity is readily available, if failures or defect rates are shown publicly using visual displays, and how readily this information is available. The central concepts here are the types of information being displayed, how broadly it is being shared, and how easy it is to access. Visibility, and the high-quality communication it enables, are key.

We hypothesized that in combination these practices increase delivery performance—and indeed they do. In fact, they also have

positive effects on team culture and performance. As shown in Figure 7.2, these Lean management practices both decrease burnout (which we discuss in Chapter 9) and lead to a more generative culture (as described in Westrum’s model in Chapter 3).



*Figure 7.2: Impacts of Lean Management Practices*

## IMPLEMENT A LIGHTWEIGHT CHANGE MANAGEMENT PROCESS

Every organization will have some kind of process for making changes to their production environments. In a startup, this change management process may be something as simple as calling over another developer to review your code before pushing a change live. In large organizations, we often see change management processes that take days or weeks, requiring each change to be reviewed by a change advisory board (CAB) external to the team in addition to team-level reviews, such as a formal code review process.

We wanted to investigate the impact of change approval processes on software delivery performance. Thus, we asked about four possible scenarios:

1. All production changes must be approved by an external body (such as a manager or CAB).
2. Only high-risk changes, such as database changes, require approval.
3. We rely on peer review to manage changes.
4. We have no change approval process.

The results were surprising. We found that approval only for high-risk changes was not correlated with software delivery performance. Teams that reported no approval process or used peer review achieved higher software delivery performance. Finally, teams that required approval by an external body achieved lower performance.

We investigated further the case of approval by an external body to see if this practice correlated with stability. We found that external approvals were negatively correlated with lead time, deployment frequency, and restore time, and had no correlation with change fail rate. In short, approval by an external body (such as a manager or CAB) *simply doesn't work* to increase the stability of production systems, measured by the time to restore service and change fail rate. However, it certainly slows things down. It is, in fact, worse than having no change approval process at all.

Our recommendation based on these results is to use a lightweight change approval process based on peer review, such as pair programming or intrateam code review, combined with a

deployment pipeline to detect and reject bad changes. This process can be used for all kinds of changes, including code, infrastructure, and database changes.

### ***What About Segregation of Duties?***

In regulated industries, segregation of duties is often required either explicitly in the wording of the regulation (for instance, in the case of PCI DSS) or by auditors. However, implementing this control does not require the use of a CAB or separate operations team. There are two mechanisms which can be effectively used to satisfy both the letter and the spirit of this control.

First, when any kind of change is committed, somebody who wasn't involved in authoring the change should review it either before or immediately following commit to version control. This can be somebody on the same team. This person should approve the change by recording their approval in a system of record such as GitHub (by approving the pull request) or a deployment pipeline tool (by approving a manual stage immediately following commit).

Second, changes should only be applied to production using a fully automated process that forms part of a deployment pipeline.<sup>1</sup> That is, no changes should be able to be made to production unless they have been committed to version control, validated by the standard build and test process, and then deployed through an automated process triggered through a deployment pipeline. As a result of implementing a deployment pipeline, auditors will have a complete record of which changes

have been applied to which environments, where they come from in version control, what tests and validations have been run against them, and who approved them and when. A deployment pipeline is, thus, particularly valuable in the context of safety-critical or highly regulated industries.

Logically, it's clear why approval by external bodies is problematic. After all, software systems are complex. Every developer has made a seemingly innocuous change that took down part of the system. What are the chances that an external body, not intimately familiar with the internals of a system, can review tens of thousands of lines of code change by potentially hundreds of engineers and accurately determine the impact on a complex production system? This idea is a form of risk management theater: we check boxes so that when something goes wrong, we can say that at least we followed the process. At best, this process only introduces time delays and handoffs.

We think that there's a place for people outside teams to do effective risk management around changes. However, this is more of a governance role than actually inspecting changes. Such teams should be monitoring delivery performance and helping teams improve it by implementing practices that are known to increase stability, quality, and speed, such as the continuous delivery and Lean management practices described in this book.

---

<sup>1</sup> For more on deployment pipelines, see <https://continuousdelivery.com/implementing/patterns/>.