The Pragmatic Programmers

Your Code as a Crime Scene

Second Edition

Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Fight Unplanned Work, the Silent Killer of Projects

Visualizing the health of your codebase offers an actionable starting point and a potential trigger for paying down technical debt. However, any organization looking to improve its delivery efficiency has to take a broader perspective. In addition to the technical improvements, you also need to reshape the engineering and collaborative strategies to ensure no new bottlenecks are introduced.

All of these changes are investments that take time, meaning we need to bring visibility to the outcome to ensure improvements have a real effect. Measuring trends in unplanned work offers a simple solution by complementing the code-level metrics with a higher-level perspective.

Unplanned work is anything you didn't anticipate or plan for, such as bug fixes, service interruptions, or flawed software designs causing excess rework. By its very nature, unplanned work leads to stress and unpredictability, transforming a company into a reactive instead of a proactive entity. In fact, in *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win [KBS18]*—a wonderful and highly recommended read—Gene Kim describes unplanned work as being "the silent killer of IT companies." Let's see how to use the concept for communicating expectations and future improvements.

Adding More People Cannot Compensate for Waste

When a company accumulates technical debt, the business increasingly experiences symptoms, commonly in the form of Jira tickets moving at a depressingly slow rate. The gut response is a cry for more developers, more testers, more of everything. Yet losing predictability is a sure sign that more people isn't the solution. In See That a Man-Month Is Still Mythical, on page ?, you will learn how adding more people will probably exacerbate the situation.

Open Up the IT Blackbox by Measuring Unplanned Work

Most organizations track unplanned work indirectly via product life-cycle management tools like Jira, Azure DevOps, or Trello. This makes it possible to calculate the ratio of planned vs. unplanned work over time. You just need to agree on which issue types represent unplanned work.

The figure on page 5 shows an example from a real-world project in crisis. Looking at the trend, you see that the nature of the delivered work has shifted over time, and the organization now spends 60 percent of its capacity on

reactive, unplanned work. There's also an overall decline in throughput, meaning less work gets completed than earlier in the year.

Focusing the presentation on trends makes the waste obvious: no organization wants to do worse today than it did yesterday. Let's put the amount of unplanned work into context by quantifying the waste.

Calculate the Untapped Capacity Tied Up in Technical Debt

We can never eliminate unplanned work, but we still need a reliable target for putting our numbers into context. A good baseline for unplanned work is 15 percent, which is what high-performing organizations achieve in terms of bug fixes. (See *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations [FHK18]*.) With the 15 percent baseline indicating the acceptable amount of unplanned work, we can now sketch out the following formula:

```
Waste (%) = UnplannedWork% – 0.15
UntappedCapacity ($) = Ndevelopers * AverageSalary * Waste
```

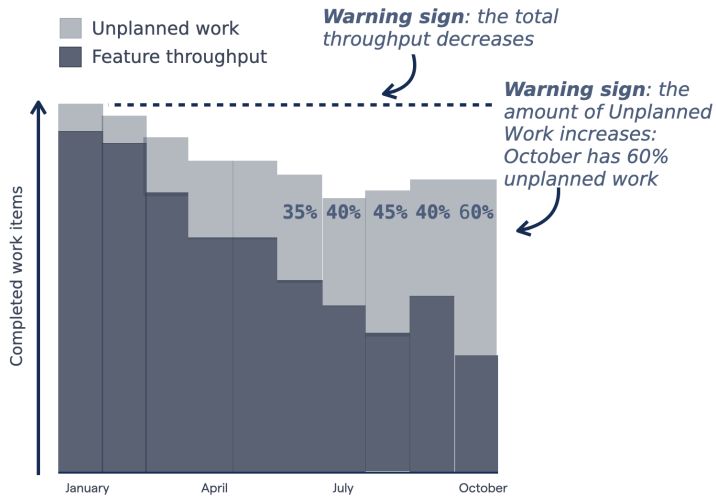
Let's run the formula using the data from the unplanned work trend for the project in crisis in the figure as an example. The figure shows that they spent roughly 60 percent on unplanned work during the last month. Assuming an average European software developer salary, we can estimate the untapped potential by filling in the numbers in our formula:

```
// Assuming an average salary of 5.000 Euros/month.
// With payroll tax and benefits, the employer pays ~7,500 Euros.
// Now, the project had 35 developers.
Waste (%) = 0.60 – 0.15 = 45%
UntappedCapacity: 35 * 7,500 * 0.45 =  118,125€ / month
```

This exercise reveals the potential when unplanned work is minimized: it would mean the equivalent of 15(!) additional full-time developers. These are not new hires; by reducing the amount of unplanned work, you free up developers to focus on actual planned work, which moves your product forward. The added bonus is that those 15 developers come with no extra coordination cost since they are already in the company. How good is that? It's hard to argue with the promise, particularly when it's your data.

Use Quality to Go Fast

Getting more done without hiring more people is a clear competitive advantage. Yet, too many companies in the industry seem to share a commonly held belief that high-quality code is expensive. You detect this mindset each time you hear a "no" as a response to a suggested technical improvement: we might not "have



time" for refactoring, test automation, architectural redesigns, and so forth—you know, the usual suspects. It's like there is a supposed tradeoff between speed and quality, where choosing one negatively influences the other.

However, as indicated by the data you met in this chapter, there doesn't seem to be such a tradeoff. In fact, the contrary seems to be true: we need quality to go fast. Use that to your advantage.

Differentiate Remediation Time from Interest Payments

The software industry has seen previous attempts at quantifying technical debt, often by (mis-) using metrics such as the *Software Maintainability Index [WS01]* or *SQALE [LC09]*. While these methods might be valuable to assess the source code itself, they lack the relevance dimension and connection to the actual business impact. Remember, the cost of technical debt is *never* the time needed to fix the code—the remediation work—but rather the continuous additional development work due to technical issues.

Measuring trends in unplanned work lets you quantify this, and combining those trends with code-health visualizations allows you to break down the impact to individual modules to make the data actionable.

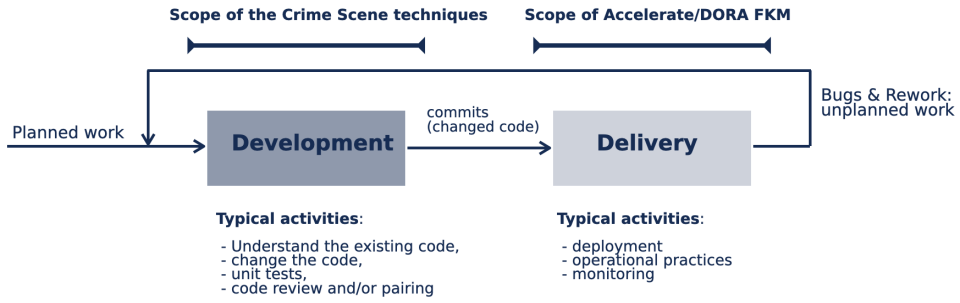
Finally, when discussing metrics and outcomes, we also need to touch on the DevOps Research & Assessment (DORA), which established the Four Key Metrics (FKM): change lead time, deployment frequency, mean time to restore,

and change fail percentage.[4] In their research, the DORA team showed that these metrics are solid leading indicators for how the organization as a whole is doing.

The DORA metrics work well with this chapter's techniques. As you see in the figure on page 7, FKM focuses on the delivery side, while this book focuses on the earlier steps in the software development cycle: the waste introduced when the code is written. At the end of the day, you need both. It's hard to go fast if you don't go well.

These days, efficient software development is a competitive advantage, enabling companies to maintain a short time-to-market with a mature product experience. Armed with a new vocabulary grounded in research, you can now assess the current waste and—most importantly—know how to communicate it to the business. From here, you're ready to expand the concepts from this first part of the book to the level of software architecture. In Part II, you'll see how the crime-scene techniques scale to the system level. But first, try the following exercises to apply what you've learned in this chapter.

4. https://www.devops-research.com/research.html


Scope of the Crime Scene techniques
Scope of Accelerate/DORA FKM
Planned work
Development
commits
(changed code)
Delivery
Bugs & Rework:
unplanned work
Typical activities:
- Understand the existing code,
- change the code,
- unit tests,
- code review and/or pairing
Typical activities:
- deployment
- operational practices
- monitoring