

SEI SERIES IN SOFTWARE ENGINEERING

Managing Technical Debt

Reducing Friction in
Software Development

Philippe Kruchten

Robert Nord

Ipek Ozkaya



Managing Technical Debt

Let's look at this chain in more detail with a simple example, based on a story introduced in Chapter 1, about a Canadian company that first developed a product for customers who speak English and then needed to make the product multilingual. This company is Atlas, the small startup, one of the three representative examples introduced in Chapter 3, "Moons of Saturn—The Crucial Role of Context."

Early in its existence, Atlas produced a demo version of its product almost overnight to show to a group of venture capital investors. A rudimentary scaffolding, called L10N (localization) and I18N (internationalization), was used in the code in lieu of proper localization and internationalization software. To appeal to another part of the Canadian population, the developers next wrote ugly code to support one other language, French, in addition to English. A few weeks later, the CEO of Atlas assured prospective Japanese customers that a Japanese version could be completed just as quickly as the French version. In fact, adding this third language proved extremely cumbersome. It required major changes in the way the code was developed. It also meant removing and redoing all the changes made to accommodate French. And it took quite some time to achieve and necessitated putting other developments on hold.

The chain of cause and effect in this case looks like this:

- **Causes:** Developers completed the first version in time for the demo under schedule pressure. They were also unfamiliar with I18N and L10N software.
- **Technical debt:** Code snippets to handle a second Latin-alphabet language were scattered all over the codebase because internationalization had not been considered as a key architectural driver and had to be retrofitted.
- **Consequences:** The code was error prone and could not support other languages, especially non-Latin-alphabet languages.
- **Symptom:** The visible consequence was a long delay to add support for a third (non-Latin-alphabet) language, when the team finally recognized the impact of the issue.

But the news was not all bad. Another consequence was that the investors were impressed by the bilingual version, and they moved forward with funding:

- **Consequence:** The company got a third round of investment from a venture capital firm (yeah!).

If members of the Atlas development team had been aware of the technical debt as it was incurred, they would have identified the risk early and could have made some

contingency plans to deal with it. It is likely the schedule would not have given them the flexibility to accommodate French debt free, but they could have begun proactively managing the debt while supporting future languages. This would have helped set expectations for a longer release when negotiating resources for the Japanese version. More often than not, technical debt is unintentional and does not become visible until much later, when consequences surface.

The first step toward recognizing technical debt is to investigate the chain of cause and effect in reverse:

Symptoms → Consequences → **Technical debt** → Causes

Using the analogy of a health issue, a physician would start from the symptoms to diagnose the problem. Similarly, you should aim to detect more consequences of technical debt, possibly less visible ones, by looking inside the system, and they will eventually point you to the development artifacts containing the debt. We call these artifacts and their associated principal and interest a *technical debt item* (refer to Chapter 2). Identifying these items aids in resolving the problem at its source rather than treating the symptoms and then seeing the problems resurface.

Pursuing this path of analysis, you next ask, “Why do we have this debt item?” The answers to this question will help you locate the *causes* of technical debt—even its root cause. While understanding causes isn’t strictly necessary for resolving the technical debt, it may provide insight into how the development environment is creating conditions for incurring technical debt. It may lead to changes in the organization to avoid generating more technical debt. We will explore causes of technical debt in more depth in Chapter 10, “What Causes Technical Debt?”

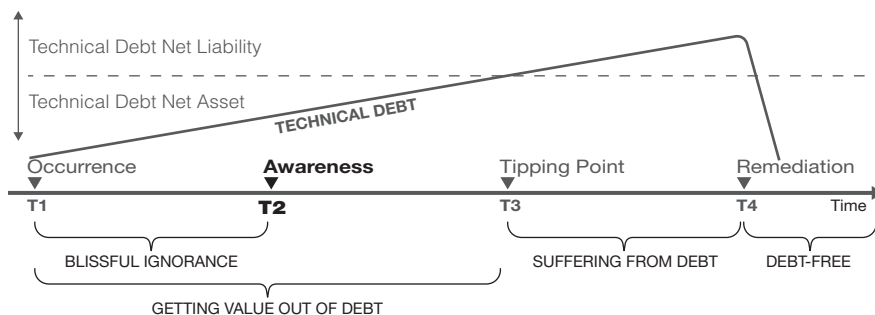


Figure 4.1 *Reaching the awareness point*

In the technical debt timeline we introduced in Chapter 2, the first goal is to reach the point of *awareness*, or knowing what technical debt you have in your system (see Figure 4.1). The technical debt item will enable you to track the debt you become aware of within your software development process so that you can estimate, discuss, and prioritize actions to take.

What Are the Visible Consequences of Technical Debt?

Some symptoms—such as the release delay faced by Team Atlas for its Japanese-language version—emerge after the entire system has been affected. They surface late in the development cycle and manifest as increased testing time, problems integrating subsystems into the rest of the system, and projects hitting major impediments that stop the release of new features. Other symptoms surface later still, during maintenance, and are reflected in increased maintainability and sustainment costs.

These symptoms are consequences of technical debt that manifest directly in the system. But consequences can reach further into the environment of which the system is a part. These consequences include an overall decline in quality that is visible to the end users and results in an increase in customer change requests or a decrease in market share as usage declines. When the consequences of debt are visible, they become easier for development teams to communicate to decision makers. Visible consequences also make it easier to get management buy-in for fixes, as Joe, a developer from Tethys, the global giant introduced in Chapter 3, summarizes:

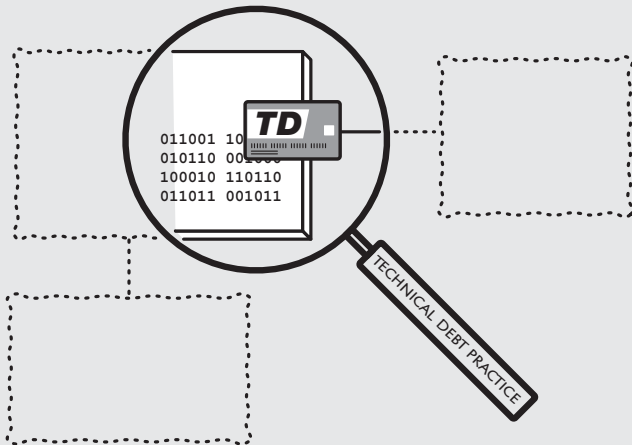
I think that it is fairly easy to convince management when performance is really bad, when they are experiencing latency, when systems stop working, and when they see exceptions on the user interface.

But it is also a risk because by the time consequences become visible, the debt may also have a higher cost of remediation.

Some symptoms of technical debt surface earlier during software development, before they affect the entire system. These include an increasing number of issues and bugs, a decreasing rate of development productivity (for example, velocity) or cumulative flow, and increasing code quality concerns (for example, cyclicity, McCabe complexity). Development teams often are aware of these symptoms, even

the debt itself, but do not have the mechanisms or incentives to communicate the issue. This is where the technical debt item can help.

Principle 4: Technical Debt Must Trace to the System



To reason about technical debt, estimate its magnitude, and offer information on which to base decisions, you must be able to anchor technical debt to explicit technical debt items that identify parts of the system: code, design, test cases, or other artifacts. A development organization also needs to recognize other forms of friction related to processes, people, and the development infrastructure. But these sources of friction are causes of technical debt; they are not the debt themselves.

When you trace technical debt to the system, start with your business context, assess artifacts across the technical debt landscape, and record the results as a technical debt description.

Writing a Technical Debt Description

A technical debt description captures *where* in the system the debt is located (the concrete system artifact) and the associated state of consequences that it causes in the system.

Recall that Atlas’s small startup team has just released the second version of its product, with the addition of French to a system that already supports English. The project team is now contemplating adding support for a third language.

A user story to describe the new feature request might take this form:

As a <stakeholder>, I want to <action with system> so that <benefit>.

For Atlas, this looks as follows:

As the Atlas Company, we want a Japanese-language version of our product so that we can increase market share and profit.

However, you need more than a user story to describe a technical debt item. You need to enhance the basic story by documenting some of the who, what, when, where, and why (also known as the *five Ws*, or 5W) to describe the problem so that you can make it visible to the entire project team and deal with it as you would any other issue on the backlog. A technical debt description is a user story that includes the five Ws that explain the associated technical debt.

Here is the 5W version of the Atlas team’s technical debt description:

As a developer (**who**), I need to pay down the debt where internationalization (**what**) is scattered over the code (**where**). The accumulating cost to add support for additional languages will soon outweigh the initial benefit of implementing the ad hoc solution of if-then-else statements for the first two releases to obtain another round of funding (**when**). There will be a long delay to support the next language, and the code will soon no longer support additional languages, especially languages using non-Latin characters (**why**).

You will need to collect your technical debt descriptions in what we call the *technical debt registry*, or *registry* for short. But you can use the same repository and tool that you are already using to manage work—your backlog.

Table 4.1 lists the essential fields to capture a technical debt item. They can easily be incorporated into your issue tracking process and technical debt registry.

Typically, to track technical debt, software development teams use whatever tool they routinely use to manage the project, such as an issue tracker system or a defect database. Most issue trackers include capabilities to create custom types and fields. We strongly recommend creating a type for technical debt items and tagging technical debt descriptions with a label, such as “techdebt,” if they are stored with user stories, defects, and other tasks.

Table 4.1 *Technical debt description*

Name	What is it? This field is a shorthand name for the technical debt item.
Summary	Where do you observe the technical debt in the affected development artifacts, and where do you expect it to accumulate?
Consequences	Why is it important to address this technical debt item? Consequences include immediate benefits and costs as well as those that accumulate later, such as additional rework and testing costs as the issue stays in the system and costs due to reduced productivity, induced defects, or loss of quality incurred by building software that depends on an element of technical debt.
Remediation approach	Describe the rework needed to eliminate the debt, if any. When should the remediation occur to reduce or eliminate the consequences?
Reporter/assignee	Who is responsible for servicing the debt? Assign a person or team. While in most cases the who aspect can be trivial, in some situations the debt resolution may need to be assigned to external parties. If remediation is significantly postponed, this field can communicate that decision.

If your team is disciplined, members can easily document the discussion of consequences and change requests as part of the detailed description field of an existing issue type. However, we often observe that software developers explain the *what* and the *where* as they incur or become aware of technical debt, but they fail miserably to highlight clearly the consequences of not fixing it, how the debt might grow over time, and a reasonable time to pay the debt if the fix must be deferred. Therefore, we recommend at a minimum creating a custom field and building the discipline to record the consequences of accumulating debt. That will help you assess how high the interest of the debt is growing. Such a simple practice has powerful operational benefits, such as retrieving all outstanding and possibly closed techdebt issues and assessing their importance and priority against the team's resources.

Table 4.2 shows the technical debt description for Atlas after the second release of its product, with the addition of French to a system that already supports English. The project team is now contemplating a third language.