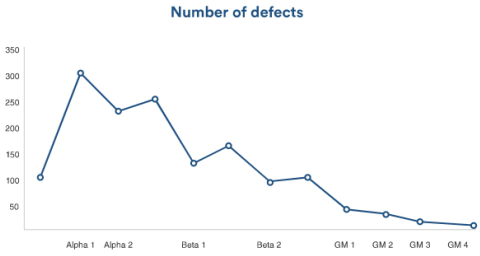
Escaping the black hole of technical debt

//TODO (Kidding!) But seriously: does your groan reflex kick in when you see that? Yeah, us too.

BY DAN RADIGAN

Traditional software programs have a phase-based approach to development: feature development, alpha, beta, and golden master (GM).



Each release begins with a phase where new features are built, and (ideally) residual issues left over from the last released are addressed (but let's be honest: this rarely happens). The development cycle reaches "alpha" when each feature is implemented and ready for testing. Beta hits when enough bugs have been fixed to enable customer feedback. Unfortunately, while the team is busy trying to fix enough bugs to reach beta, new bugs appear. It's a chronic case of whack-a-mole: fix one bug, and two more pop up. Finally, the release hits the golden master milestone when there are zero open bugs. "Zero open bugs" is usually achieved by fixing some known issues, and deferring the rest (most?) to the next release.

Constantly procrastinating on bugs that need to be fixed is a dangerous way to make software. As the bug count grows, tackling it becomes increasingly daunting–resulting in a vicious death-spiral of technical debt. To make matters worse, schedules get derailed because coding around the bugs slows down development. Meanwhile, customers are experiencing death by a thousand cuts caused by un-fixed defects. And indeed, some of them will leave you.

There has to be a better way.

Reducing technical debt through agile

Agile bakes quality into the iterative development approach so the team can maintain a consistent level quality release after release. If a feature isn't up to snuff, it doesn't ship. Find that hard to believe? Well there's a trick: defining, or *redefining*, the definition of "done."

For traditional teams, "done" means good enough for QA to begin. The problem with that definition is that bugs creep in early in the release cycle–and continue to creep in. So by the time QA gets their hands on it, the product is saddled with layers upon layers of defects. Agile teams, however, define "done" as ready to release, which means developers don't move on to the next story or feature until their current item is practically in customers' hands. To speed things along, they use techniques like feature branching workflows, automated testing, and continuous integration throughout the development cycle.

> Preventing technical debt is what allows development to be agile in the long run.

The main branch of the code base should always be ready to ship. That's priority number one. So new features begin their lives on a task branch containing code for the feature itself, plus it's automated tests. Once the feature is complete, and the automated tests pass, the branch can

then be merged up into main. Because the quality bar is always fixed (and fixed high), technical debt stays under control.

For many organizations this is a huge cultural change. With agile, the focus away from schedules and towards high-quality, demonstrable software. The product owner is empowered to focus the team on the most valuable work first, reducing the scope of the release instead of compromising on quality.

Because we can't forget: the longer bugs linger, the more painful they are to fix.

Taming your team's debt

If you're working with legacy code, chances are you've inherited some technical debt. The following topics will help you tame existing debt, and enable your team to focus on the fun stuff like new feature development.

Define it

Sometimes developers and product managers disagree about what constitutes technical debt. So let's put the controversy to bed right here:

> Technical debt is the difference between what was promised and what was actually delivered.

This includes any technical shortcuts made to meet delivery deadlines!

There's a temptation on the development side to characterize architectural work as technical debt. It may or may not be, depending on the nature of the change (e.g., replacing a shortcut with the "real" solution vs. splitting a monolithic code base into microservices). On the other side, product management often feels more urgency around building new features than fixing bugs or slow performance. To avoid either side becoming jaded about the other party's opinion, everyone needs to understand the distinction between technical debt, desired architectural changes in the code base, and new features. Clear communication between development and product management is critical in prioritizing the backlog and evolving the code base.

> PRO TIP:
>
> **Prioritize technical debt in sprint planning just like normal feature work. Don't hide it in a separate backlog or issue tracker.**

Beware of testing sprints and tasks

Fight the urge to compromise the definition of done by adding a separate testing task to the original user story. It's too easy to defer them and only invites technical debt. If testing isn't done as part of the original story or bug fix, the original story or bug fix isn't done. Maintain a strict definition of done in your program and ensure it includes complete automated testing. Nothing saps the team's agility more than manual testing and a buggy code base.

Automate bugs away

When someone discovers a bug in the software, take the time to add an automated test that demonstrates it. Once the bug is fixed, rerun the test to ensure it passes. This is the core of test-driven development, a time-honored methodology for maintaining quality in agile development.

Where to start?

Changing the team's philosophy (and that of the team's stakeholders) on how to manage technical debt isn't easy. Business needs sometimes cut development time short in order to get to market sooner. With that in mind, let's recap some action items for taming technical debt:

- Educate the product owner on the true cost of technical debt. Ensure story point values are accurate for future stories that require resolution of existing technical debt.
- Modularize your architecture, and take a firm stance on

technical debt in new components or libraries in the application. As the team and business see the agility in these new components, they will naturally want to extend those practices to other parts of the code.

- Write automated tests! Nothing prevents bugs better than automated tests and continuous integration. When a new bug is found, write a new test to reproduce it and then fix the issue. If that bug ever resurfaces, the automated test will catch it before customers do.

Remember, technical debt is a reality for all software teams. Nobody avoids it entirely–the main thing is to keep it from spiraling out of control.

SHARE THIS ARTICLE

DAN RADIGAN

Agile has had a huge impact on me both professionally and personally as I've learned the best experiences are agile, both in code and in life. You'll often find me at the intersection of technology, photography, and motorcycling.

TUTORIAL

Learn scrum with Jira Software

A step-by-step guide on how to drive a scrum project, prioritize and organize your backlog into sprints, run the scrum ceremonies and more, all in Jira.

Try this tutorial →

ARTICLE

Get better quality with agile testing practices

QA teams are responsible for executing test plans. With agile testing they can sustainably deliver new features with quality. Learn best practices here.

Read this article →