Engineering higher quality through agile testing practices

There's still a need for manual testing–but not in the way you might think!

BY DAN RADIGAN

Waterfall project management separates development and testing into two different steps: developers build a feature and then "throw it over the wall" to the quality assurance team (QA) for testing. The QA team writes and executes detailed test plans. They also file defects when painstakingly checking for regressions in existing features that may have been caused by new work.

Many teams using these waterfall or other traditional testing models find that as the product grows, the amount of testing grows exponentially–and QA invariably struggles to keep up. Project owners face an unwelcome choice: delay the release, or skimp on testing. (I'll give you one guess as to which option wins 99% of the time.) In the mean time, development has moved onto something else. So not only is technical debt mounting, but addressing each defect requires an expensive context switch between two parts of the code base. Insult, meet injury.

To make matters worse, QA teams are traditionally rewarded according to how many bugs they find, which puts developers on the defensive. What if there was a better way for both developers and QA to reduce the number of bugs in the code while also eliminating those painful trade-offs project owners have to make? Wouldn't it create better all-around software?

Enter agile and DevOps testing.

Moving from traditional to agile testing methods

The goal of agile and DevOps teams is to sustainably deliver new features with quality. However, traditional testing methodologies simply don't fit into an agile or DevOps framework. The pace of development requires a new approach to ensuring quality in each build. At Atlassian, the way we test is agile. Take a detailed look at our testing approach with Penny Wyatt, Jira Software's Senior QA Team Lead.

> Let's be clear: scripted manual testing is technical debt.

Much like compounding credit card debt, it starts with a small amount of pain, but snowballs quickly–and saps the team of critical agility. To combat snowballing technical debt, at Atlassian we empower (nay: expect) our developers to be great champions for quality. We believe that developers bring key skills that help drive quality into the product:

- Developers are great at solving problems with code.
- Developers that write their own tests are more vested in fixing them when they fail.
- Developers who understand the feature requirements and their testing implications generally write better code.

We believe each user story in the backlog requires both feature code and automated test code. Although some teams assign the developers the feature code while the test team takes on automated testing, we find it's more effective to have a single engineer deliver the complete set.

> PRO TIP:
>
> **Treat bugs in new features and regressions in existing features differently. If a bug surfaces during development, take the time to understand the mistake, fix it, and move on. If a regression appears (i.e., something worked before but doesn't anymore), then it's likely to reappear. Create an automated test to protect against that regression in the future.**

This model doesn't mean developers work alone. It's important to have QA engineers on the team as well. QA brings an important perspective to the development of a feature, and good QA engineers know where bugs usually hide and can advise developers on probable "gotchas."

Human touch through exploratory testing

On our development teams, QA team members pair with developers in exploratory testing, a valuable practice during development for fending off more serious bugs. Much like code review, we've seen testing knowledge transfer across the development team because of this. When developers become better testers, better code is delivered the first time.

> Exploratory testing makes the code, and the team, stronger.

But isn't exploratory testing manual testing? Nope. At least not in the same sense as manual regression testing. Exploratory testing is a risk-based, critical thinking approach to testing that enables the person testing to use their knowledge of risks, implementation details, and the customers' needs. Knowing these things earlier in the testing process allows the developer or QA engineer to find issues rapidly and comprehensively, without the need for scripted test cases, detailed test plans, or requirements. We find it's much more effective than traditional manual testing, because we can take insights from exploratory testing sessions back to the original code and automated tests. Exploratory testing also teaches us about the experience of using the feature in a way that scripted testing doesn't.

Maintaining quality involves a blend of exploratory and automated testing. As new features are developed, exploratory testing ensures that new code meets the quality standard in a broader sense than automated tests alone. This includes ease of use, pleasing visual design, and overall usefulness of the feature in addition to the robust protections against regressions that automated testing provides.

Change can be hard–really hard

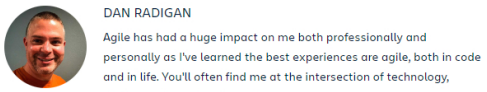
I'll leave you with a personal anecdote that nicely summarizes my journey with agile testing. I remember managing an engineering team early in my career that had strong resistance to writing automated tests, because "that work was for QA". After several iterations of buggy code and hearing all the reasons why automated testing would slow the team, I put my foot down: all new code had to be proven by automated tests.

After a single iteration, code started to improve. And the developer who was most adamantly against writing tests turned out to be the one who sprung into action when a test failed! Over the next few iterations the automated tests grew, scaled across browsers, and made our development culture better. Sure, getting a feature out the door took longer. But bugs and rework dropped significantly, saving us huge amounts of time in the end.

Change is rarely easy. But like most things worthwhile, if you can buckle down and create new patterns for yourself,

the only question you'll be left with is "Why didn't we do this sooner?!"

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

Incident response for agile development

When bugs in production, incidents, and downtime happen, learn how to build user trust by applying agile values to your incident response.

Read this article →

Sign up for more agile articles and tutorials.

Email

email@example.com

Subscribe

Up Next

Incident response →