JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Motivation Visibility, and Unit Testing

ARTICLE

By Steve Berczuk | May 19, 2010

SUMMARY

Steve Berczuk has always been interested in organizational patterns and recently found himself thinking about motivation. Agile practices, both technical and organizational, build a framework which makes having the right amount of collaboration and feedback possible. But there's a bootstrapping process: How do you get people to start doing the practices, especially technical ones, such as unit testing?

I've always been interested in organizational patterns (such as those in **Organizational Patterns of Agile Software Development**). I've recently found myself thinking a lot about motivation. I'm now reading **Drive: The Surprising Truth About What Motivates Us** and just finished **Rob Austin's book on performance measurement**. Being the parent of a three year old, I'm finding more and more that "because I said so, and I'm right" isn't too effective at home.

My interests in motivation are closely related to my interest in writing software. Writing software is partially a technical problem about frameworks, coding, and the like, but the harder (and perhaps more interesting) problem is how to get a group of people working together towards a common goal. Agile practices, both technical and organizational, build a framework which makes having the right amount of collaboration and feedback possible. But there's a bootstrapping process: How do you get people to start doing the practices, especially technical ones, such as unit testing?

In an ideal world, everyone will know how to write unit tests, understand their value, and want to write them. In an organization transitioning to agile having all three parts in place is not a given. Most people understand why unit tests are useful, in principle. The problem is the execution. In my experience, there are two reasons people who claim that that *want* to write unit tests give for *not* writing them:

- They are too *hard*. The overhead of the test can make the cost of developing a feature excessive.
- They are too *easy*. Some functionality appears to be trivial, so why would we want to test it.

We use cookies to ensure you get the best experience on our website.

Accept

Read More

business critical that could be quickly validated by visual inspection ("make the background color of the home screen orange") may well be not worth the effort. The problem is that most people have bad intuitions about where the lines line until they start practicing the skill of unit testing. Most teams need experience with testing to get a true feel of what is a trivial test, and what is a seemingly trivial test that can unmask a major problem

Rather than frame the testing challenge with the default being the old way of not testing:

> *Write a test when it makes sense.*

Change your perspective to the default being to test:

> *Write a test unless you can explain why you did not.*

The key to this approach is to make encourage people to think through their rationale for not testing. There are a few ways to do this, but one approach is to tie the explanation mechanism into something every developer works with every day: your source code repository. Have the team agree that, in addition to the source files, every commit will have a change to a test or a rationale for why you didn't write a test. For example if I do a commit without a test I could write:

> *ISSUE-23: Fixed the spelling of the company name. NO TEST: it was a typo.*

or

> *ISSUE-26: Fixed the rendering mechanism. NO TEST: we don't have a good framework for testing this sort of thing*

or even

> *ISSUE-28: Fixed a serious logic issue. NO TEST: I didn't feel like writing one.*

By developing a team agreement to add tests or explain why not, you are starting with a small change of behavior that paves the way for a greater change based on understanding. Even if a message like the last one is acceptable, many will be uncomfortable admitting to laziness, and think harder for a reason. By reviewing the commit messages later on you can get a sense of impediments to testing (technology, organizational, or attitude), and use that data in a **retrospective** to decide how to improve.

By being creative you can help people on your team understand the value of process changes, and start a conversation about how to evolve practices to suit your team.

Topics: unit testing,

We use cookies to ensure you get the best experience on our website.

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

UPCOMING WEBINAR

LETS HANG!

Log In or Register to post comments

We use cookies to ensure you get the best experience on our website.

0 comments

Join the crew for FREE! Access exclusive content, contribute, and more.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.