Feedback without Fear

by Stephen P. Berczuk

Agile software development favors forward progress in small steps with the opportunity to make corrections along the way. While we may not know everything at the beginning, we know enough to move forward a bit at a time. In a successful agile project everyone acknowledges that there is uncertainty and that the uncertainty increases as we look further ahead. To make the appropriate corrections, we need to have useful feedback in place.

Unfortunately, the words "I want to give you some feedback" tend to elicit groans rather than enthusiasm. A phrase that elicits almost as many groans is "software configuration management." Many teams don't understand that their build environments and software configuration management (SCM) systems can be extremely useful tools to provide the feedback that an agile process needs.

Simple Tools for Feedback

Every software development project should have two basic tools in place for feedback: a build environment and a version management system. A build environment (with the appropriate automated tests) tells you if the code base is still good, and a version management system can tell you that things are changing and how. It's hard to argue with what working (or broken) software in the form of a continuous integration build is telling you: If you have tests that define your requirements, you know if you are meeting your criteria for success. A passing or failing test tells you a lot more than a vague statement about progress.

Feedback from the Build

With continuous integration tools (CruiseControl and Continuum are two freely available examples) it is straightforward to set up a process that tells you something about the quality of the code. In a matter of minutes one can install CruiseControl, for example, and have a process that will:

- Monitor the version management system for changes
- Build a component when it detects a change
- Run any automated tests you have in place; and if you use a build tool like Maven, you can configure it to fail the build if certain metrics such as coding style are violated or if unit test coverage metrics are not met
- Send email to team members with the success or failure of the build, including information on what the last changes were and who made them

So why doesn't everyone use the build to provide feedback on the state of their applications?

One issue is the lack of automated tests. Writing tests takes time and not all architectures are suited for simple developer testing or automated integration testing. Rather than giving up, you should use this as an opportunity to find out as much as you can. Does everything build and integrate? Even a test that simply starts up an application without errors provides useful information. I have worked on projects that actually had unit tests, but what kept giving us problems were errors in some configuration files that were read at startup. Until we knew to write tests for that, we spent more time than I'd like to admit tracking down those errors. (The version management system helped us with that task!) Once you have these basics in place, you can explore how to make testing more effective.

A common excuse is that the tests that exist aren't really good tests of end-to-end functionality, so why bother? While automated integration tests are good to have, no test is too trivial. As much as agile is about combining small steps to build a useful system, any application is the sum of its parts. I've been in situations where a team spent hours tracking down a problem that was caused by an incorrect Java `hashCode()` method. People on the team thought that writing a test of `hashCode()` was too trivial. You need to start somewhere, and any information is better than no information.

Feedback from Your Version Management System

Builds tell you how things are now. Your SCM system can help you identify how you got there and give you the ability to recover from errors.

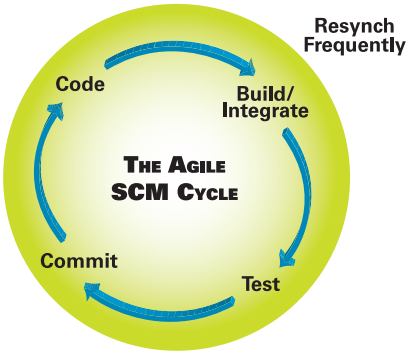
Most continuous integration systems provide you with information from the SCM system about what changed between the last build and the current build. When everything works, this is often just interesting information. When something breaks, it gives you a tool to track down what might have caused the problem. It's important to realize that if a change appears in a broken build, the change itself might not be the problem. That change might have exposed a mistake lurking from earlier in the day. But the SCM system can give you a place to start to evaluate why something broke and where to look to fix it.

Another valuable service an SCM system gives you is reproducibility. If someone on your team is encountering a problem, or he can't reproduce a problem in his workspace, the SCM system can help identify whether the workspace contains the correct components, thus providing an explanation when someone exclaims that it "works for me!"

When working with their version management systems, teams often develop overly complex codeline strategies. While branching is a useful tool, it is meant to provide for isolated variations. There are times when the isolation makes sense, but in many cases teams use isolation to hide problems and avoid potentially scary feedback. To get the best feedback from your SCM system, use the simplest model that works, and be sure that the team understands the codeline and branching strategy.

Avoid the Blame Game

When using your build and SCM systems as feedback mechanisms, it is important to remember that the end goal is improving quality—not casting blame or resting on laurels. You need to use the results of the failed builds to work together as a team to fix the problem; perhaps the last person who makes a change is the right person to fix the problem, but not always. And a series of passing tests should cause you to wonder if there is more that you can check for in your build and more that you can automate, or if there is a way to have the build and tests run more quickly.



Once you realize that the goal of the build feedback process is to know as much as you can—as soon as you can—about the state of your codebase, these and other reasons not to use your build for feedback become insignificant.

Conclusions

When you are trying to solve a problem, it's best to step back and focus on why you are doing what you are doing and understand what the problem really is. The answer is often simpler than you think. Feedback in a form that makes it easy to self-correct is essential to an agile process. There are simple steps you can take now to give your team the feedback it needs to be more successful. {end}

Stephen P. Berczuk, a Certified Scrum Master and co-author of the book Software Configuration Management Patterns: Effective Teamwork, Practical Integration, *is a senior software engineer at FAST. Steve can be contacted at steve@berczuk.com. His Web site is www.berczuk.com.*

Index to Advertisers

Better Software (USPS: 019-578, ISSN: 1532-3579) is published twelve times per year. Subscription rate is US $75 per year. A US $35 shipping charge is incurred for all non-US addresses. Payments to Software Quality Engineering must be made in US funds drawn from a US bank. For more information, contact info@bettersoftware.com or call (800) 450-7854. Back issues may be purchased for $15 per issue (plus shipping). Volume discounts available.

Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to *Better Software*, 330 Corporate Way, Suite 300, Orange Park, FL 32073, info@bettersoftware.com.