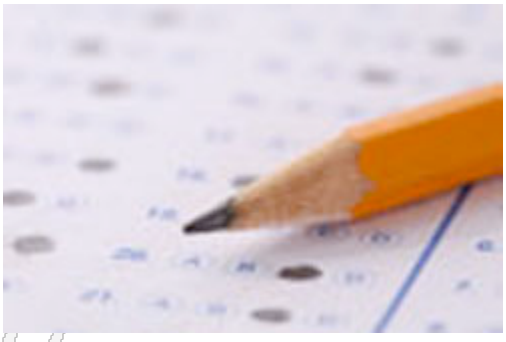
Agile SCM is Testing

Tuesday, 16 October 2007



The Software Configuration Management Patterns include a number of patterns around testing, and we discuss testing in this column occasionally. From time to time we hear the question: "what does testing have to with software configuration management, anyway?" We claim that testing is essential for Agile CM Environments, and that agile CM environments are built on testing. To explain this idea we need to talk about what SCM is and how testing helps.

A software configuration management system is in place to manage change. In a 1990 Software Engineering Institute Technical Report Susan Dart said

> *The goals of using CM are to ensure the integrity of a product and to make its evolution more manageable.*

The traditional approaches to software change management focuses on the software lifecycle and identifying and controlling what changes happened in what context. Identification of the facts of changes is important. What really matters to many stakeholders is less the fact of the change, but the impact of the change to them. People care that their system works as expected, the functionality was added as desired and not removed accidentally. We want to ensure that the value of our configurations is increasing over time rather than decreasing! To verify these things you need to test the working application. In this sense you can't fulfill the goals of SCM without testing.

As a system evolves you want to know at what point risks increase. The more frequent (and automated) the testing the easier makes it is to identify the point in time when your system is in a configuration that does not work, which is where something like continuous integration comes in. The recent book Continuous Integration: Improving Software Quality and Reducing Risk by Paul Duvall discusses how a CI system is a central part of the software quality lifecycle as you can test not just for functionality, but also for various metrics.

We will now describe the tradeoffs between and identification-based approach to SCM and a verification based approach.

Stability and Progress
CM environments balance:

- Stability - how certain you can be that the code works at any given point
- Progress - how quickly a codeline evolves to encompass new features or fixes.

How to set the balance varies depending on the needs of the project. Stability is always important since without stability it's difficult, if not impossible, to make progress. Stable in this sense means that little changes, but by this definition, "stable" can become "stale" rather quickly. A more useful definition of stable is that the quality of the code is staying as good as it was. With this sense of "stable" positive change can happen.

One way to ensure stability is to add controls and processes to ensure quality, but it is easy to make the controls interfere with the business of an organization: delivering new functionality. The classic example of this is requiring changes go through extensive (manual) review before being worked on and then requiring extensive (and time-intensive) testing before making a change. The idea behind this is well meaning: software is complicated and we don't know the effect of a change, so let's be very cautious in making changes. These rules will improve stability, at least on the surface but at a great risk to your rate of progress. In our previous article, The Illusion of Control we discuss that it is often better to have more visibility into effects of a change rather than attempt to guarantee that a change will be "safe." (For more on how to emphasize transparency over tracability see our September 2007 article Lean Traceability: a smattering of strategies and solutions.

Another approach is to understand that it's extremely difficult to understand the effect of a change, and instead change the criteria for approving a change to be that the change did what it was expected to do, and did not break any existing functionality. You can do this by having a codeline policy that requires that new code have:

- New Unit Tests

- Pass a workspace build. including unit tests
- Pass an integration build (including all unit and integration tests)

Each of these rules can also be validated through an automated process; test coverage tools can allow you "test" whether new code has unit tests, for example, testing not only functional compliance (the code still works based on the tests) but process compliance (the metrics that we consider important are also met). Continuous Integration: Improving Software Quality and Reducing Risk has excellent practical advice on how to use your build process to measure various quality and policy metrics.

Another difference between this point of view and traditional SCM is that traditional SCM is often "event-based", focused on baselines, individual changes, etc. Lean/Agile SCM is focused on managing the flow of change across the value stream. In an event-based model the fact that a developer made a change is of primary interest, and our infrastructure is focused on tracking (and perhaps preventing) the developers from making a change. In the model we're discussing here the item of interest is the impact that the developer's change had on the system, and we can use criteria like code quality to initiate action, rollback changes, etc. The difference is one of priorities and focus. We care about tracking the various events, as they allow us to recover when things start going in the wrong direction. But we want to report on the impacts, not simply the events.

Tests
Much has been written about the different kinds of tests: Functional Tests, Integration Tests, Unit Tests and who write them (developers, QA Engineers, etc), so we won't cover that here. It is important to understand that there are many places in the Software Development Ecosystem Timeline where testing happens and each has a impact on SCM.

- During Coding: software developers write unit tests for any changes/additions that they are making, and frequently run the unit tests for other parts of the code to ensure that their change did not break anything. This might also be a good time to extend the functional test suite. When a developer feels that their work is ready to share, she updates her workspace, does a final build and runs the unit test suite.
- Once code is submitted an automated integration build is run. This build might run all the unit tests as well as any integration tests.
- Periodically (nightly or more frequently as possible) longer running automated regression or functional tests are fun.
- As new featured appear on the scene, manual testing can happen for those features that do not yet have automated tests.

Roles
The "SCM is Testing" position also frustrates people because it seems to be placing an QA function (Testing) in the hands of another team (Release Engineering). To be able to successfully respond to change you need to forget about those boundaries, and think of SCM as being an element (perhaps a central element) of the software development environment.

As we mentioned above, there is a sense in some circles that testing is not relevant to the SCM community because testing is not part of build management or release engineering. The problem with this idea is that if SCM is about ensuring the integrity of the product, what other mechanisms do we have to do this? While we often speak in favor of cross functional teams where people do not have traditional roles and everyone has a broad skill set, it is likely that many organizations have some sense of boundaries.

- The Software Developer
  - is responsible for writing unit tests for any code she touches and for running any tests available so as to verify the stability of the codeline.
  - might also work with the QA Engineer to write functional tests.
  - might help maintain build scripts for modules they understand architecturally
- The QA Engineer
  - can help with defining functional and integration testing
  - can look at incorporating test coverage reporting into the automated suite - to ensure that code is appropriately exercised by those tests, and to maintain the desired levels of test coverage for new functionality
- The Release Engineer
  - can enable the development team to do local builds
  - help define criteria for acceptance at various levels

Process and Workflow Testing
Many companies use workflow and tools to help automate their process, e.g. having defects being reported and going through a certain lifecycle with different sign-off points and different levels of authority. Agile teams tend to keep these processes as simple as possible - allowing people to make decisions flexibly based on their own judgement. But what happens if you are in a situation where this is not deemed sufficient? Don't forget that changing a process requires testing to ensure that the process now does

something slightly different. It is very common to find organisations making process changes on the "live" system in a rather uncontrolled manner.

So, treat your workflow process as something that needs:

- change control and configuration management - can you reliably report on what was changed and when? Can you roll back changes if they didn't work?
- testing - as for code - can you automate this and make sure the workflow still works and a minor change hasn't broken something else?
- release - do you have a test environment so that changes can be made offline, tested and then applied to the live environment

These things need to be taken into account when choosing the tool you will be using for your process - what reporting requirements are necessary, are there any licensing implications for test environments?

How to get there

While not part of the traditional definition of SCM, testing is an enabler for an agile SCM environment. Rather than controlling risk by slowing change, you control risk by monitoring the state of your codeline after each change.

Testing and test driven development are not things that happen naturally alas, and the reasons for this could be the subject of another article. Here are some steps that you can take:

- Make sure that running tests is part of your build (even if tests don't exist yet).
- Make passing tests part of the criteria for a good build
- Include code coverage metrics in your build reporting. Consider failing builds if test coverage goes down (which implies that code was added without tests)
- Start writing tests. (where to start, strategy)

Writing tests where there are none is challenging, but necessary.

In Conclusion
Rather than being besides the point of SCM, an appropriate testing strategy is what enables an agile SCM environment. To be more agile, you need to avoid the "silo" based perspective of development, SCM, and test being different disciplines with interfaces, and you think about how the processes in one part of your development ecosystem affects what you can do in the others. An effective change management system is built upon good testing practices, and effective change management is one of the things that Agile SCM environments are about.

Brad Appleton *is an enterprise SCM/ALM solution architect for a Fortune 100 technology company. He is co-author of* ***Software Configuration Management Patterns: Effective Teamwork, Practical Integration,*** *the "Agile SCM" column in CMCrossroads.com's CM Journal, and a former section editor for The C++ Report. Since 1987, Brad has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. He holds an M.S. in Software Engineering and a B.S. in Computer Science and Mathematics. You can reach Brad by email at brad@bradapp.net*

Robert Cowham *has been in software development for over 20 years in roles ranging from programming to project management. He continues his involvement in development projects but spends most of his time on SCM Consultancy and Training. He is the Chair of the Configuration Management Specialist Group of the British Computer Society, has a BSc in Computer Science from Edinburgh University and is a Chartered Engineer (CEng MBCS CITP). You can contact him at rc@vaccaperna.co.uk*

Steve Berczuk *develops software applications at Fast Search and Transfer in Boston, MA. He has been developing object-oriented software applications since 1989, often as part of geographically distributed teams. In addition to developing software he helps teams use Software Configuration Management effectively in their development process. Steve is co-author of the book* ***Software Configuration Management Patterns: Effective Teamwork, Practical Integration****. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at steve@berczuk.com.*

Trackback(0)

TrackBack URI for this entry

Comments (2)

Subscribe to this comment's feed

Moo: ...

Nice article to highlight the importance of testing in the context of SCM. Would it be possible for you guys to provide a BibTeX or similar citation for this or future articles?

vote up vote down report abuse 1

February 13, 2008
Votes: +0

Atul Kulkarni: ...

Is the best article, to know the basics of Agile and SCM. Also helps to know the process and workflow of Agile SCM is testing.

vote up vote down report abuse 2

October 23, 2007
Votes: +0

Write comment

Last Updated (Tuesday, 01 April 2008)