Agile SCM: Basics For Small Teams

As much as software developers are stereotyped are solitary coders, software development is a collaborative activity. Communication among team members is essential in ensuring working continuously working software. And working software is what makes communication with stakeholders easier. You can show the state of your application rather than explain progress in terms of more abstract concepts. Your SCM system (and processes) are an essential part how you communicate both in and about code between developers and to stakeholders.

Communication gets more complex the larger the group of people. Small teams have the advantage of being able to be aware of what each person is doing, and of having shared product vision. Small teams, being small, are also resource constrained, so you want to maximize time spent delivering value, and minimize the time spent in process overhead.

The Short Lists

Given the time constraints of small teams we'll be brief and list the essential items for success with SCM in tools and processes, grouped into general priorities.

Really Basic SCM

These items might seem too basic to mention, but we'd be remiss not to mention them:

- Use a version management system (SCM Repository) as the only way you share code. There is no excuse for sharing code any other way. An SCM system gives you the ability to track changes, and have a central place to backup and recover the business asset that is your code.
- Create a process that enables one to get a workspace up and running quickly. A new person joining the team should be able to start with a set of tools, and then check out source code, run the

application, and make a change on their first day. The process of making this possible will also make your deployment model more robust.
- Use a continuous integration (CI) environment. Having your code in a repository is good, but only in the sense that it is usable by others when they check out the code. A CI environment gives you a sanity check that your source code builds, which is the minimal criteria for it being usable. Also, enabling a CI environment means that you need to have a build process that is somewhat portable and consistent.

Keeping things working

The basic list will allow you to collaborate effectively and deliver features quickly, but you may still find yourself stumbling unless you add some practices to eliminate distractions:

- Create automated tests and run them as part of your build. Compiling, which necessary, is not sufficient for keeping code working. Automated testing, while adding time, will pay off with added robustness of code and the ability to make changes reliably.
- Work off of a single code line to start. Branch only once you have a delivered codeline to maintain, that you expect is stable. Each branch is a parallel line of development, and parallel work means a distraction from your main work.
- Deploy often. The real test of whether your SCM process is helping you to deliver product is how effectively you can deploy. Don't wait til the end of a release to think about deployment practices. Start deploying your application the first day. This guards against problems only being discovered late in the process.
- Automate your deployments. The more automated your process is the easier it is to deploy more frequently. This becomes a virtuous cycle (of positive feedback).

Trace and Improve

The previous lists will get you most of the technical issues, but there are still things you can do improve how you collaborate.

- Identify your commits with meaningful messages. Be explicit about

the reason for a change in terms of business goals. Keep your commits and consistent and cohesive - avoid mixing changes to refactor existing code that doesn't change the functionality with changes to add new functionality - split those into separate commits. If you use an issue tracking system, associate each commit with an issue number. Otherwise, refer to a feature or user story. While the code can often speak for itself, context is helpful.

- Think, and Review. Since every team is different, and teams themselves change over time, it's important to review how you are working, and figure out which practices are working for you and which are not. Retrospectives are a key agile development practice, but they aren't just for agile developers. Periodically make a list of the things about your SCM and Release process that seemed to be working well, and which need improvement.

Recap and Resources

While in an ideal world you'd do everything on this list on day 1, you might have practical roadblock to doing that, so we broke the list into parts....

To learn more, here are some resources we like:
Practices to use for small team scm, and their rationale: SCM Patterns Book(Berczuk, S. P. and B. Appleton (2003). Software configuration management patterns : effective teamwork, practical integration. Boston, Addison-Wesley.)

Deployment: (Jez Humble's book) Humble, J. and D. Farley (2010). Continuous delivery : reliable software releases through build, test, and deployment automation. Upper Saddle River, NJ, Addison-Wesley.

Retrospectives: Agile Project Retrospectives: Derby, E. and D. Larsen (2006). Agile retrospectives : making good teams great. Raleigh, NC, Pragmatic Bookshelf.

Unit Testing: XUnit Patterns(Meszaros, G. (2007). xUnit test patterns : refactoring test code. Upper Saddle River, NJ, Addison-Wesley.)
CI Practices and approaches: Continuous Integration. (Duvall, P. M., S. Matyas, et al. (2007). Continuous integration : improving software

quality and reducing risk. Upper Saddle River, NJ, Addison-Wesley.)

You'll find that some of these books overlap in content a bit. This is because it takes an assortment of practices to collaborate effectively. On a small team you have the advantage of easier communication. Use that to your advantage to simplify your process, but don't underestimate the value of process in making communication easier.

About the Authors Brad Appleton is an enterprise SCM solution architect for a Fortune 100 technology company. Currently he helps projects and teams adopt and apply agile development & SCM practices. Brad also author's the Agile CM Environments blog, and is co-author of Software Configuration Management Patterns: Effective Teamwork, Practical Integration, the "Agile SCM" column in CMCrossroads.com's CM Journal, is a regular contributor to "The Agile Journal", and is 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 for VIZIM Worldwide. He is the Chair of the Configuration Management Specialist Group of the British Computer Society, has a BSc inComputer Science from Edinburgh University and is a Chartered Engineer (CEngMBCS CITP). You can reach him by email at robert@vizim.com

Steve Berczuk is a consultant and developer who works with Agile teams. He has over 20 years experience developing application, 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 and a Certified ScrumMaster. 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

Excerpted from *Agile SCM: Basics for Small Teams*

http://www.cmcrossroads.com/agile-scm/13844-agile-scm-basics-for-small-teams

---

Readability — An Arc90 Laboratory Experiment

http://lab.arc90.com/experiments/readability