Crossroads News
A Monthly Publication for Software and CM Professionals

Agile SCM – It's All Related

Steve Berczuk, Steve Konieczka, Brad Appleton – July 2003

This month we will talk about how using some basic patterns can help you build a software configuration management process that works well with your agile development environment. We will discuss how Codeline Policy, Private Workspace, Smoke Test, Private System Build, Integration Build, Unit Test, and Regression Test all work together to enable you to maintain an Active Development Line.

When you work in a release engineering or SCM role you can start talking and thinking in terms of "Software Configuration Management" as an end in itself. If you are developing software, you often ignore SCM, treating it as something necessary that hangs out in the background, or, if it is visible, you think of SCM as being "the version control tool." Neither view of the world will give you the most effective environment. It is important for everyone to remember that SCM is a part of the environment where you develop code. It's easy to forget this when, as a practical matter, your day-to-day work has you working with things of immediate import, no matter how much you want to think in global terms. Since we tend to specialize, much of the literature on SCM is focused on the practices of SCM without necessarily fitting the practices into context.

When Brad and Steve were working on the pattern language for agile software configuration management, which evolved into the book: *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, we had this in mind. Patterns and pattern languages, by their nature, help you to place things in context. In this article we will give you an overview of part of our SCM pattern language with the goal of showing you how to present SCM practices in the context of the entire development process.

Patterns and Pattern Languages

There has been much written about patterns and pattern languages, and it is a topic that can take a while to master. For the purposes of this discussion there are a few things to know. In the book *A Timeless Way of Building* Christopher Alexander describes a pattern as something that "describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." A shorter definition is "a rule which describes what you have to do to generate the entity which it defines."

Alexander describes the importance of context in *A Pattern Language*: "… every pattern we define must be formulated in the form of a rule which establishes a relationship between a context, a system of forces which arise in that context, and a configuration which allows these forces to resolve themselves in that context."

There are a number of collections of best practices, etc, but we often keep hidden the context of how these practices fit in with our larger goals. Brad and Steve wrote a pattern language for agile software development that is organized around SCM practices. This pattern language (which you can learn more about at http://www.scmpatterns.comT) defines some core development practices related to SCM. The benefit of this approach is that you are more likely to have increased success when applying the patterns and those who are performing SCM related tasks on a daily basis will have a better understanding of when to do the tasks.

To learn more about patterns and pattern languages, some good starting points are:

- Brad Appleton's Patterns FAQ(http://www.cmcrossroads.com/bradapp/docs/patterns-nutshell.html)
- Some of the books by Christopher Alexander et al: *A Timeless Way of Building*, *The Oregon Experiment*, and *A Pattern Language*
- The Hillside Group's web site: www.hillside.net

SCM in Context

The pattern language for agile SCM starts with the basic idea of an *Active Development Line*. In essence, an *Active Development Line* is a codeline that has a *Codeline Policy* that permits frequent check-ins. If you have worked with a team developing software you might be concerned more about someone breaking the build because of a careless change than with how often code changes. While delayed integration does cause problems, your concerns about breaking the build will be valid. For *Active Development Line* to work you need to have other patterns in place to give you some confidence that the rapid changes will not harm to codeline quality too much.

To address the issues of build integrity, you want to be able to build the code before checking it in as well as test it. To build the code you need to have a place to put the code, a *Private Workspace* gives you a place to put the code, do your development, testing etc. To verify that the code builds, developers can do a *Private System Build* pre-check-in. And once the developers check the code in, it is built in integration environment using an *Integration Build*.

A *Private System Build* is a build that is as close as possible to the *Integration Build* (and official system build). Setting up a *Private System Build* can help you avoid issues caused by developers having a different approach to building that the *Integration Build*.

After each developer checks the code in, you will still need to integrate the code in a central place. To build the code in an integration workspace, set up an *Integration Build* which is the “official” build.

The first step in addressing testing issues is to define a *Smoke Test* that developers run pre-check-in. A *Smoke Test* is a basic test of the important aspects of the system. The *Smoke Test* that you run before checking in code covers essential functionality, and runs quickly. “Essential” and “quickly” are both concepts that you have to decide how to define for your project, but if your goal is an agile codeline, you don’t want a 1 hour smoke test. This leaves the issue of How do I really test the code? *Unit Test* and *Regression Test* address this.

A Unit Test is a low level, detailed test of interfaces that developers run on the code that they are working with. You can write your unit tests using a unit testing framework such as JUnit for Java code, cppUnit for C++ code, etc, but that is not essential to have unit tests.

A *Regression Test* is an exhaustive system level test that verifies that known problem have not re-appeared in the code. Construct unit tests based on known issues, or by an analysis of likely failure points. Always run the *Regression Test* after the *Integration Build*, or before checking code in to a codeline that you need to be stable.

Lastly, the *Codeline Policy* is a description of what procedures to follow before checking code in. A *Release Line* may have a more restrictive policy than an *Active Development Line*, for example.

The figure below shows how the patterns fit together. An arrow from *pattern A* to *pattern B* means, in effect, that pattern B is necessary to ‘complete’ pattern A. This is a structural relationship and does not have any defined implications about the sequence in which the patterns are implemented. The key thing is that you can’t just say “I want an Active Development Line” and change the codeline policy appropriately.

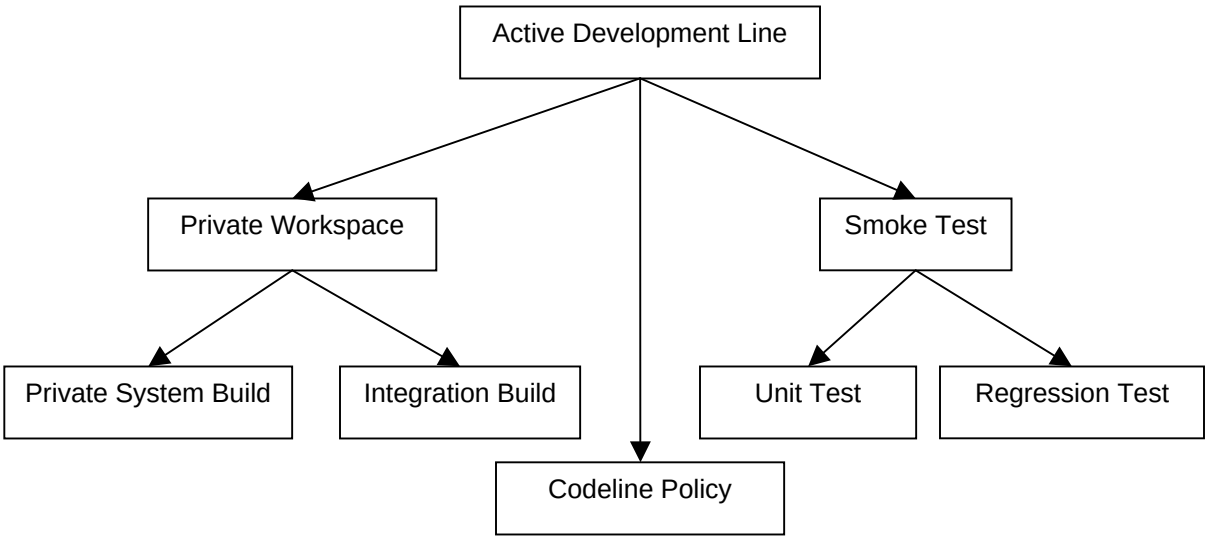

Figure 1 Part of the Pattern Language

Putting it All Together

If you read this far you may be thinking "*Codeline Policy*, and build issues, are SCM, but isn't testing QA?" In a sense that is correct, but the fact that they are separated into two different groups is an organizational, and perhaps historical, artifact. In Extreme Programming teams developers write tests, and use version control, in addition to coding. There is a role for testing experts, ideally with that role fully integrated into the team, either as someone who helps customers write acceptance tests or as someone on the development team who helps developers with their testing technique. Likewise, there is a role for software configuration management in an agile project, and indeed, all projects would benefit by a more integrated view of SCM.

In addition to abstract benefits of understanding the global value of SCM solutions, implementing SCM processes with this in mind will make it easier for the entire team to embrace the SCM and appreciate the value that it adds.

Conclusion

Often we focus on our particular discipline and lose track of the goal of a software project: To build software that serves a customer's needs. Testing, SCM, as well as coding should be part of every developer's toolkit. Some will know more than others about a given discipline, but the disciplines need to work together well.

Next month we will discuss Agile Change Management.

Acknowledgements

Ron Jeffries provided input into this article. To learn more about Extreme Programming visit www.xprogramming.com.

Brad Appleton is co-author of H**Software Configuration Management Patterns: Effective Teamwork, Practical Integration**H. He has been a software developer since 1987 and has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. In addition to SCM, Brad is well versed in agile development, and cofounded the Chicago Agile Development and Chicago Patterns Groups. 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*

Steve Berczuk is an Independent consultant who 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*. His web site is www.berczuk.com

Steve Konieczka is President and Chief Operating Officer of SCM Labs, a leading Software Configuration Management solutions provider. An IT consultant for 14 years, Steve understands the challenges IT organizations face in change management. He has helped shape companies' methodologies for creating and implementing effective SCM solutions for local and national clients. Steve is a member of Young Entrepreneurs Organization and serves on the board of the Association for Configuration and Data Management (ACDM). He holds a Bachelor of Science in Computer Information Systems from Colorado State University. You can reach Steve at *steve@scmlabs.com*