**Crossroads News**

**A Monthly Publication for
Software and CM Professionals**

# Agile SCM – Patterns and Software Configuration Management

**Steve Berczuk, Brad Appleton and Steve Konieczka – April 2004**

*Abstract*

Patterns and pattern languages are tools that you can use to help your team be more effective and more agile. Patterns can lead to robust, effective solutions because the solutions that patterns lead you to take the environment into account, and solve problems in a way that makes the system work better. This article will show you how you can use existing patterns to improve your SCM process. It will also help you to understand where existing patterns and pattern languages have gaps.

## Introduction

Since this issue of CM Crossroads focuses on Software Configuration Management (SCM) Patterns and two of the contributors of the Agile SCM column are the authors of a book on an SCM Pattern language, it seemed appropriate to focus this month's article more on SCM Patterns than on Agile SCM. The themes or agility and patterns are closely related; the appropriate use of Patterns and Pattern Languages can help you to be more agile. This month we will provide a quick overview of what Patterns and SCM Patterns are with enough information to get you oriented. We'll also provide a list of resources to find to learn more about patterns and pattern languages.

If you already know what SCM Patterns are about, this article will provide you with some ideas about how to get your organization to adopt them. If you are new to SCM Patterns (or patterns of any sort), this article will get you started. This column may start off a bit more theoretical than many of our other columns; if that is not what you prefer, hang on for a bit. There will be practical advice. When we wrote SCM Patterns we wanted to encourage people to understand what they were trying to accomplish first, and apply practices and tools later.

## Introducing Patterns and Adding Value

For "Patterns" to be useful, you need to think of them as more than just a magic phrase, or silver bullet that everyone will understand and appreciate. By using patterns and pattern languages you can add value to your organization by improving the development process as a whole, rather than simply fixing immediate problems. Patterns also provide for a common vocabulary for SCM concepts, which will facilitate communication in your organization.

To use "patterns" in your organization, your audience need not be versed in what patterns are, or in the specific patterns that you wish to use for you to leverage their power. In some organizations the people in a team will be aware of what patterns are, and they may be familiar with a certain class of patterns. This is not the case in most organizations. As with any attempt on process improvement how, much value you get from patterns depends on how you introduce the concepts as much as the values of the concepts alone. Often the various aspects of software development such as "coding," "configuration management," "testing" — to name just a few — are treated as separate steps which concern different people. This leads a sense of competition rather than cooperation.  A pattern approach will help you and your team to understand how SCM practices fit within the development process, thus encouraging everyone to move towards a common goal: higher quality software delivered more effectively.

To benefit from SCM patterns you need to introduce them into your organization in a way that will have an impact. Walking into in a room and speaking of "Patterns" will not get you very far unless the people in the group understand what patterns are and have some clue about the particular patterns that you are discussing. They may hear the word "pattern" and have some reaction to it (positive or negative) and leave without understanding the real goal: working more effectively, much like the dog in the favorite Gary Larsen *Far Side* which has a dog hearing "Ginger *blah blah blah* Ginger" rather than the complete message. This can be frustrating, especially if you understand how to implement an effective SCM process, and if you have thought a bit about how patterns work with SCM.

You are more likely trying to influence people who are so focused on their immediate tasks that may not have even heard of Patterns, much less, SCM Patterns. So if we're going to talk about how patterns help you to be more effective and more agile, we may want to consider thinking of how you can use patterns effectively without spending too much time on introducing the concept of patterns. After all, in the end people usually want a better software development process, not an education in patterns or other concepts. (Though through the education, they may be able to better understand how to fix their own process)

If *you* understand the patterns they can provide a framework for helping to explain the value in ways that your audience can grasp. For example, many readers of CM Crossroads *know* the value of *Private Workspaces* (for example) and how they fit into the development process, so I hope that the *Private Workspace* pattern in the book can provide an outline for you to argue why a team should use private workspaces (or anything similar) Patterns, be they from *Software Configuration Management Patterns*, or from other sources. Patterns are most effective in the hands of a "champion" who can interpret the patterns and translate the concepts into a mechanism that works for the existing culture. While there is benefit to adopting the pattern vocabulary, it is more important to have your organization adopt the principles from the patterns.

## The "What" of Patterns and SCM

Patterns differ from other forms of describing best practices in that patterns make the relationship between practices explicit. Patterns build upon each other, and when they form a coherent unit, they form a Pattern Language, which is a guide to building something. Often you are faced with a problem for which a best practice sounds like a solution. You are then faced with questions like:

- How do I put this practice in place?
- What are the organizational and process prerequisites for this practice to work?
- What are the implementation considerations specific to my organizations tools and SCM environment?

A pattern contains the following basic parts:

- A name,[1] which describes what the pattern helps you build. The name also forms the basic for a common vocabulary. You can talk about a Task Branch in an unambiguous way if you all agree that Task Branch is what the pattern Task Branch describes.[2] Tools often have their own vocabulary and you may need to translate between the terminology that your organization and tool[3] uses.
- A Context, which describes when it makes sense to build the thing that the pattern describes. Rather than say "Build a task branch" we need to know if the environment is one where a Task Branch will be helpful and useful. In the SCM Pattern language, the Task Branch makes sense in the context of an Active Development Line. In a perfect world, we could express the context in

[1] Names are often noun phrases to emphasize that patterns are things. Some have objected to the idea of expressing SCM Process Concepts as "things" rather than "tasks." Either way works, but there are some benefits to the consistent thing model.

[2] Ideally, the name is based on some common usage, so rather than "creating" a vocabulary, the name helps to establish meaning within a common vocabulary.

[3] We are working with some vendors to come up with a mapping between the tool vocabulary and the SCM Patterns where the tool supports the concept, but under another name.

terms of other patterns. In some cases no one has written the pattern and we settle for a prose description of your situation.

- A description of the problem, both a summary and a detailed description explaining the tradeoffs that you need to consider when solving this problem.
- A solution, that shows how to use the patterns described to solve your problem, and how the pattern solves your problem. Typically you will see a short solution and as well as a longer description with implementation details.

Keep in mind throughout this discussion that a prerequisite for a pattern is that it describes a proven good practice (within the context). The form does not make it a pattern.

A pattern collection or pattern language is a guide to helping you to build a certain kind of thing. Various pattern languages may share some elements, but a pattern language for an Agile SCM environment will be different than one for a more controlled environment, and trying to steer an organization in an inappropriate direction will cause problems. The Pattern Language in the book *Software Configuration Management Patterns* is about building a software CM environment where your want to respond rapidly (agilely) to change. Fortunately, this is a common requirement at the heart of many processes, and the SCM Pattern language addresses how to separate the parts of your codebase that need to change rapidly, from those that need to be stable.

## Who Can Introduce Patterns

Process Improvement is often thought of as being the stuff of "initiatives" and "working groups" but most organizations should be trying to improve in small ways every day. Understanding what patterns should be in an environment is part of that process.

When I started out as a software developer I worked with a team of people who taught me that how we do things and why we do things as developers is as important as what we do. In other words: process isn't only the concern of senior developers. Everyone should care about finding ways to work better. Of course, the more influence you have over the team, the more you should care, since you'll have a bigger impact on the bottom line, but there are things that everyone can do to make their own work better and perhaps lead by influence.[4]

Perhaps only a few should thing globally. Everyone should act locally. For example, any developer should be able to add tests for their code, independent of the team's global policy.

## Goals and Intentions of the SCM Pattern Language

Perhaps the common definition of "SCM Patterns" is the patterns in the book *Software Configuration Management Patterns*, along with, perhaps, the patterns in *Streamed Lines*.[5] A pattern language guides you along the process of building a thing, the SCM Pattern language in *Software Configuration Management Patterns* is not a guide to building every SCM process. The patterns in *Software Configuration Management Patterns* work best in an agile environment. I could argue that most teams should be more agile than they are now, and could be so without sacrificing anything. Many restrictive (or non-agile) processes seem to have their origins in lack of trust, or an unreasonable fear of risk, rather than reasonable business goals.

[4] The book *Becoming A Technical Leader* is an great resource for learning how to do this. See the Resources section of this article for publication information.

[5] See http://www.cmcrossroads.com/bradapp/acme/branching/

You are the one who best understands what your team is doing now and how it needs to improve.

## The Role of Tools

The first question I get about SCM is often “What tool should I use?” Like all things, tools influence the way that you work. If you have a tool that makes something easy you may do something one way. Without that tool another approach may be almost as good. My advice, which is not often well received, is to understand what your work process is, what you want it to be, what your constraints are, and then pick a tool that supports them. Then you can think about whether you should use different tools.

The pattern language in *Software Configuration Management Patterns* does not care about tools beyond having some sort of version control system and some sort of build process tool. Of course, some tools make implementing some patterns more transparent than other tools do.

## Applying the SCM Pattern Language for Process Improvement

A frequent complaint is that teams spend too much time stabilizing their codeline before releases, resulting in developers idling while the team integrate (and thus delays that cascade into subsequent releases). One approach to resolving this situation is creating a release branch and dividing the team’s effort into stabilization and new release activity. This does not always work as expected if the release branch takes a long time to reach “release quality.” Time spent waiting for the release to be ready is replaced by time spent on merging release line fixes into the main line. The SCM Pattern Language provides a map to making resolving some of these issues.[6]

A patterns approach would analyze how the problems relate and provide solutions that build upon each other. If your team has frequent releases you would attempt to focus most of your work on a single *Active Development Line*, which would have a *Codeline Policy* that allows for frequent check-ins, but only after they run a suite of *Smoke Tests* and *Unit Tests* in their *Private Workspaces*. Developers will also be expected to do *Private System Builds* to provide added assurance that their check-in will not break the build. Later on an *Integration Build* will build everything, and perform extensive *Regression Tests*. By the time you are ready to start a *Release Line*, the code should be fairly stable, and you should not need to perform complicated merges between codelines.

While having a *Release Line* (applying a single pattern) relieved some pain, applying the *Release Line* pattern in context gave you a greater benefit.

## The Missing Patterns

The SCM Pattern Language as it appears in the book *Software Configuration Management Patterns*, and the other patterns that are in the *Streamed Lines* Pattern Language and other places do not solve every problem the development team faces. Often, the most important patterns to document and disseminate are the ones that the "expert practitioners" use regularly and talk about, but never take the time to formally write-up in a book or paper that will be readily accessible to users who need it most. Here are some areas for which additional patterns would be useful. Some of these come from the list of patterns that Brad and Steve didn’t include in the book for a variety of reasons. Many come from questions that we’ve been asked, either on the SCM Patterns discussion list, or in private.

We invite you to think about practices that seem to work well in your environment, practices that seem to solve problems in a good way and that balance many difficult tradeoffs. Maybe you know the basis for another pattern that hasn’t been documented yet.

[6] We’re using the SCM Pattern Language as an example; there are other pattern languages out there.

- **Build Management** - e.g., for using or structuring make/ANT files and their corresponding "rules" (maintaining build dependencies, building all or part of a system, supporting multi-platform or component-based building, parallelizing builds or using a "build-ring", standard sets of Make/ANT targets (e.g., clean, install, update, test, docs, etc.)
- **"Physical" software design patterns** (a.k.a. code architecture) - for example, best practices for how to store and organize assets under configuration management across one or more repositories, and across directory-tree structures within a repository in order to assist not just software design/architecture, or to minimize build-dependencies or build-times, or assist traceability and/or configuration auditing
- **Change Control** - setting up, chartering, and running and facilitating a CCB (see patterns from Ellen Gottesdiener's book *Requirements by Collaboration*[7]), deciding policies for obtaining stake-holder buy-in (see Agile SCM column for Aug 2003 CMCrossroads)
- **Configuration Audit/Review** – conducting or automating audits and reviews of builds and baselines
- **Configuration Status Accounting/Reporting** – tracking and managing the status of requests, changes, and baselines the face of multiple releases and/or variants, change propagation and coordination with multiple codelines and/or sites and/or vendors/suppliers, devising and implementing promotion lifecycle models
- **Distributed and/or Multi-Site SCM,** including integration coordination/synchronization "topologies" for distributed parallel development
- **Multi-Component and/or Product-Family SCM** - effective strategies for working with multiple versions of multiple components being integrated into a system, and/or for families of components with subsets of components and component versions being combined and configured to build/package/release multiple products in a product-line
- **SCM Solution selection/deployment/architecture** – plotting the particulars of how SCM will operate within your organization, defining an overall process framework and “governance”, evaluating SCM tools and technology, administering and deploying/upgrading tools and processes across a network of servers and repositories and workspaces, coordinating and synchronizing upgrades of repositories and workspaces
- **Database CM** – CM of databases, stored procedures, database schemas and views, queries and reports; configuration and use of database workspaces and production “sandboxes” and database “code integration.” The Agile SCM Column in the January 2004 issue of CM Crossroads is a starting point for some of these issues.
- **Web SCM** – configuration and content management of websites and web software, including all artifacts (code, HTML/XML files, XML schemas and ontologies, JavaBeans and EJBs, managing JAR/WAR/EAR files)
- **Service-Oriented SCM** – SCM for service-oriented architecture, and service-oriented architecture of SCM itself (e.g., CM Services model of Dart&Wallnau and mentioned by Louis Taborda in his SCM futures/predictions article earlier this year)
- **Enterprise CM** – managing change across the enterprise, CM of enterprise architecture, CM with and for business processes, workflow, B2B and EAI

[7]E. Gottesdiener, *Requirements by Collaboration : Workshops for Defining Needs*. Boston: Addison-Wesley, 2002.

## *The Pattern "Language" Angle*

As previously mentioned, patterns describe individual solutions to a recurring problem in a context. A pattern language is more like a seasoned "trail guide" thru a set (or family) of related patterns that build upon one another. They help "connect the dots" to realize a more complete overall SCM domain solution. Documenting more proven SCM patterns is important, but so is being able to successfully select from and navigate through a growing body of SCM patterns. In addition to inviting you to document and disseminate more SCM patterns, we also heartily encourage you to help others see how patterns work together. Possible topics include:

- *Connecting and inter-relating patterns*- taking some existing "named" best-practices and showing how they work together to create a larger solution (where to start, where to go next, what factors to consider in choosing the next stop on the destination). One possible example would be to take the patterns in the *Software Configuration Management Patterns*, and in *AntiPatterns and Patterns in Software Configuration Management* [8] and relating them all together to "connect the dots" between them.

- *Sharing experiences with power of pattern names* to convey an SCM concept in a tool-independent way, and/or with a "suite" of patterns to create a "shared language" in one's own "shop" for discussing and disseminating SCM problems/solutions and process

- *Relating patterns or pattern-languages across domains* - for example, taking a set of SCM patterns and relating them to a set of Software Testing Patterns, or a set of Project Management Patterns, or a set of Requirements Management/Analysis Patterns, or Software Architecture Patterns.

- *Rationale for using patterns and pattern format* (and pattern languages) as an effective means of naming and sharing knowledge of SCM best-practices (as opposed to the approach currently taken by BOKs such as PMBOK and SWEBOK)

## Resources to Learn More

While you don't need to understand patterns to get value from a book on patterns, you can get more value from working with patterns if you understand what they are about.

To learn more about patterns and pattern languages:

- The Hillside Group, a nonprofit corporation dedicated to improving human communication about computers by encouraging people to codify common programming and design practice, has a page of resources about patterns: http://www.hillside.net/patterns. The page has links to many resources about patterns, as well to links to existing software patterns and pattern languages.

- Brad wrote an Introduction to Patterns available from: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html which has pointers to many excellent books about patterns, both in software and in the world of building architecture, where the idea of patterns originated.

To learn more about SCM Patterns:

[8] W. J. Brown, H. W. McCormick, and S. W. Thomas, *AntiPatterns and Patterns in Software Configuration Management*. New York: Wiley, 1999.

- Steve and Brad's book: *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*[9] is a published pattern language about SCM patterns that focus on teams that want to use SCM practices effectively.
- http://www.scmpatterns.com has links to resources for those interested in learning more about SCM Patterns, including a reference card for the patterns: http://www.scmpatterns.com/book/refcard.html

For help with introducing patterns:

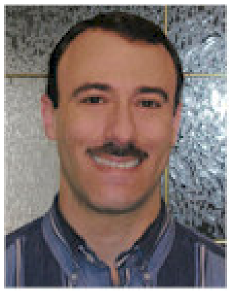
- *Fear Less: Introducing New Ideas into Organizations* by Mary Lynn Manns and Linda Rising, from Addison-Wesley, October 2004. There is a draft of some of the material on the web.
- *Becoming A Technical Leader.* (New York, NY: Dorset House, 1986) by Gerry Weinberg is an excellent book that discusses how "leadership" happens at many levels in an organization.

## Conclusion

There are many teams that have problems with basic SCM issues, where SCM isn't facilitating communication and teamwork, but doing more the opposite. The problem isn't a lack of tools; there are many good tools, both free and commercial. The problem is that people don't understand how SCM practices fit into their environment. Pattern thinking is one way to help people understand how everything is related.

[9] S. P. Berczuk and B. Appleton, *Software Configuration Management Patterns : Effective Teamwork, Practical Integration*. Boston, MA: Addison-Wesley, 2003.

**Brad Appleton** is co-author of **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**. 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** 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 *mailto:steve@scmlabs.com*