Crossroads News
A Monthly Publication for
Software and CM Professionals

Agile SCM – Characteristics of an Agile SCM Solution

Steve Konieczka, Brad Appleton, Steve Berczuk – June 2003

In the last two columns we spoke about the people and process/practice characteristics of agile development, and of an agile SCM environment. This month, we would like to focus a bit on the characteristics of SCM solutions and tool/environments that are necessary to achieve these goals, particularly about agile SCM striving to be as transparent and "frictionless", automating as much as possible.

Around the turn of the Century, (year 2k), we began seeing software projects that were attempting to try a new development approach called Agile. Most were implementing Extreme Programming, and their reasons were to reduce cost, improve software quality, better manage project risk, and to offer the business quicker functionality in the form of frequent releases incorporating smaller sets of change.

Since then I've been fascinated with the agile movement. I began to study the SCM requirements created out of the agile movement. This list of features and characteristics is not intended to be a complete list of everything required, just the ones that are most important and somewhat different from traditional SCM.

Research was based on feedback from customers, participation on agile News Groups, involvement with projects implementing agile methods and discussions with individuals in local user groups. In this research I discovered 6 characteristics of SCM solutions that stand out as major discussion points for the agile community when it comes to SCM. These 6 Characteristics or Features will provide the main focus of this article.

1 - Frictionless Code Integration

Agile projects absolutely require Frictionless Code Integration, which is defined in many ways in the SCM industry and accomplished by using a number of different features currently available on the market. In short:

- It's the freedom to change code in your private workspace without being interrupted by the SCM tool. (Optimistic Locking, Concurrent Development, etc)
- It's being able to work on your private workspace in a disconnected mode (off the LAN), and having the tools to reconnect your workspace later easily and quickly.
- It's having tools available to quickly and easily synchronize your private workspace with the repository. Synchronizing involves the ability to automatically or interactively merge file changes.
- When branching is required for the project, it's the ability to integrate changes from one branch to another in a quick and easy manner.

There are many tools on the market that provide flexible locking models. Some agile projects, in the interest of keeping things simple, prefer to have no branching, which eliminates the need for branch merging. It really depends on how the project structures its releases. When a project desires branching, it's important for an agile SCM solution to quickly and easily integrate changes from one branch to another.

Remember the agile project's needs typically require off-line development (say on the bus on the way home), and thus the need to easily reconnect their private workspace to the repository. This feature typically is a "refresh" type of function that resets the states of each component in the workspace, giving the Developer a sense for how much integration is required before committing code.

2- Refactoring Support

Agile projects change file names and directory locations quite often as they constantly simplify their code. This practice of simplifying and making the code cleaner is called Refactoring. SCM tools have traditionally struggled when supporting changing file and directory names and locations. People outside of the SCM community oftentimes don't understand the challenges of tracking file history with prior releases of code as the files are changing.

Agile SCM enables Developers to make quick and easy changes to their code with little "friction", while keeping track of file and change history. The SCM tool should offer a solution that feels as simple as Windows Explorer or any other operating system CRUD operation. Agile SCM solutions must enable the Developer to change their file names anytime they want and present them with an easy to understand interface to connect file history when and where appropriate.

Refactoring Support is probably one of the biggest challenges facing the SCM Tool market. There are tool vendors that are working hard to better support Refactoring. I predict that over the next several years, Refactoring Support in the SCM Tools market it will be relatively commonplace.

3- Continuous Integration

One of the four core values of agile software development is, "Working Code over Comprehensive Documentation." Working code is a highly valued concept for any agile project. A way to ensure that the code in the repository is in working condition is to perform periodic and automated Integration builds and tests.

Since we know that the Development Team values working code, it should be as simple as requiring that they update their workspace, build the application and run their tests against it prior to committing their changes to the repository. This approach, while important to the entire development process, works most of the time and is relatively inexpensive. However, there's no guardrails to getting non-working code into the repository, and there are holes to relying solely on this development process.

There are cases when the application will unit test fine in a Developer's environment but nowhere else. I call this the, "It Works on My Machine" scenario. This can occur when a Developer writes to a new library or piece of code that is not yet checked into the repository. When they build and test the application in their development environment, everything works fine – on their machine. When they check in their changes but forget to check in the new library or miss a file change, it stops working for everyone else.

The Continuous Integration approach constantly measures the quality of the application and offers independent feedback to the Development Team on a regular basis. An implementation may initiate the Integration Build upon a check-in event. The build occurs on a separate integration machine with a known configuration. Then a suite of automated tests is run against the resulting build. The results are formatted into an email and sent to the Development Team. Other implementations can be kicked off periodically (say every 60 minutes) if there have been changes to the repository since the last Integration Build. As you've probably noticed, a Continuous Integration approach requires a fully automated build process.

4- Automated Builds

Automated builds seem to be one of the most misunderstood concepts in what we do. I break them down into three types of builds: Developer Build [Private System Build], Release Build [No match in the Pattern Language, but similar to Integration Build] and Integration Build [Integration Build]. The Developer Build is what the Developer uses in their private workspace to compile the application for unit testing. The Release Build, which uses the same process as the Developer Build, also includes the labeling of the release, the check in of targets, creation of BOMs, other reporting, etc. The Integration Build, which also uses the same process as the Developer Build, hooks in a test harness but doesn't typically apply any labels. The section refers to automating the Developer Build. The benefits of investing in a fully automated build system for

your application are numerous indeed. Automated builds provide speed, consistency, predictability, quality and traceability.

Developer builds occur constantly throughout the development process by Developers in order to test their changes. Let's look at a typical project of 5 developers over a one-month period. It is not uncommon for a developer to do 5-10 builds in their private workspace per day to verify their work. Over a one-month period, the Developers on this team will have performed 825 builds of the software - and that's just the first month. The point is that this is activity is done over and over again and should be automated to save time and ensure consistency.

Build systems that have manual steps have no traceability except through separate documentation, which is rarely accurate. By having the build fully automated and part of the project, you are tracking all changes to the build system along side changes to the application, therefore providing build system traceability.

5- Simplicity and Flexibility

When it comes to tools for an agile project, the agile community is very clear to say that simplicity and flexibility are mandatory. There are many tools on the market that implement processes not compatible with the project's approach, not compatible with other tools being used, etc. The cost of implementing these types of solutions can be very expensive and most commonly show up as slowed productivity for the development team.

Agile incorporates "iterative" development, which by nature requires built in flexibility. I recently heard a good analogy for "iterative" from Mike Cohn (XP Denver member, www.mountaingoatsoftware.com) that compared iterative to playing golf. When you tee off, you shouldn't have to analyze in detail the breaks of the green - you should be considering wind, general direction of the green, hazards, etc. You are trying to get the ball closer to the green to set yourself up for your next shot. You then analyze the detail that is necessary for your next shot, and so on until you put the ball in the cup. Waterfall says that you analyze ALL details before you tee off, put the plan together and stick to the plan. In real life, we get smarter as we go and should set ourselves up to adjust easily as needed. Therefore, we need processes and tools available that allow us to modify our plans as we go.

Traditional SCM says that we put together a rigorous SCM Plan up-front that dictates what artifacts we'll create, how we'll build the system, all dependencies that we have, etc. To change any of these requires an act of Congress along with a documentation effort, and likely unnecessary approvals from folks who aren't "in the know." For an agile project, we really don't need to know all of these things when we're teeing off. So, the initial SCM Plan for an agile project should document only those things that are necessary for that first shot as well as more static characteristics of the application. We'll have time to refine things when we walk down and assess our lie. This is where flexibility is so important. SCM solutions must provide traceability on artifacts even though they're being renamed, deleted or copied, or new file types added. They must provide a frictionless way to associate file changes to "business level units of change" called Change Requests (CRs), User Stories, Requirements, etc. As discussed earlier, SCM solutions should put up inexpensive guardrails instead of forcing expensive process. Tools that enforce process excessively require significant administrative overhead, decrease the team's velocity and increase the time and resources required to make adjustments to the process.

I'm not saying that to be flexible, the tool shouldn't enforce process. The SCM solution you implement to support your agile project should be set up to support a changing scope and changing process – encourage that change and keep track of the artifacts as they pass through the changing process. It's a difficult challenge, but one worth taking on.

6- Low cost Administration

Agile projects are about small teams working on applications, one iteration at a time. These teams rarely are greater than 15 members in size. Therefore, most agile projects are not going to have dedicated SCM Managers assigned to the project. If you don't have a resource dedicated to administering the SCM solution, the solution itself must be somewhat simple and inexpensive to maintain.

Agile projects have little tolerance for spending limited resources on administrative tasks that don't directly contribute to working code. The agile community is constantly looking to focus it's limited resources on productive development of working code, so if they can accomplish that while spending less on administering their SCM solution, they'll do that. When implementing your SCM solution, consider the cost of administration over time to you and to the development team. It might be a deciding factor.

Conclusion

When I step away and look at an ideal agile SCM solution from a high level perspective, I see a solution that supports the development process, embraces change to the process while keeping track of changes to the project artifacts. It provides tools and resources that allow the developer to maintain application configuration during the development process as well as being able to document that configuration once development is finished.

Traditional SCM typically approaches the development group from a more "controlling" perspective, while agile SCM really attempts to understand the needs of the development group and marrying those needs in a solution that also satisfies the needs of the SCM Manager. Agendas can sometimes get pitted against one another, but squint just a little and those agendas should begin to appear headed for common goals such as producing quality software that meets the needs of the business, on time and on budget, while still maintaining well understood and persistent application configuration.

Next month in Agile SCM, Steve Berczuk will describe patterns that help with building an Agile SCM Environment.

The Co-Authors

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 *steve@scmlabs.com*