Agile Product & Project Management Advisory Service
Executive Update Vol. 12, No. 10

SCM and Build:
Keys to an Agile Lifecycle

by Steve Berczuk

Software configuration management (SCM) and build management are misunderstood disciplines. Many organizations have practices in these areas that either hinder productivity or sacrifice too much traceability in the name of improving short-term productivity. SCM and build, when done correctly, can provide a framework that allows a team to develop code quickly. However, it's also possible to establish practices that can hinder your development team. This *Executive Update* discusses the importance of the build and SCM processes and what you need to know to help these processes be effective.

SCM AND BUILD AT THE CENTER

SCM and build management do not, at first, seem as complicated, interesting, or challenging as other aspects of software development. But SCM and build practices are a central part of a project's daily activities and provide a key mechanism for communication among stakeholders. Together, they provide for immediate visibility into the state of your project as well as tools to help you manage change.

SCM and build processes affect both how developers work every day, and how the organization manages its application lifecycle and release process. Since they are central to coordination and productivity, SCM and build present many interesting challenges. Thus, these processes deserve serious consideration from management and project stakeholders to ensure that they help and not hinder the team's productivity.

While each has distinct concerns, SCM and build management are closely related disciplines. SCM is about providing the tools and mechanisms to determine what functionality the code provides, while the build process provides validation that what you expected to see in the code is actually implemented. Taken together, the build and SCM functions also provide an essential communication mechanism among project stakeholders.

DEFINING SCM AND BUILD

The most visible aspect of SCM is version management. The first thing many think about when considering SCM is version management tools. While tools can make work simpler or more difficult, the tool you use should be a secondary consideration to understanding the SCM approach that works best for your team.

While not the whole of SCM, version management practices are central to fulfilling SCM's key functions, which, according to Susan Dart, formerly of SEI, are:[1]

- **Configuration identification.** Determining what source code the team is working with can be achieved through a combination of build and version management processes.
- **Configuration control.** Managing the release of a product and changing it in a consistent way throughout the lifecycle primarily involves your product management and issue-tracking processes.
- **Status accounting.** Recording and reporting the status of components and change requests and gathering vital statistics about components in the product spans all of your processes, though you can generate many useful statistics about code and costs of change as part of the build process.
- **Audit and review.** Validating the completeness of a product deliverable typically involves collaboration between your version management, issue-tracking, and testing processes.
- **Build management.** Controlling build configurations with your build and version management systems allow for repeatable build and a robust "manufacturing" (as Dart refers to it) process.
- **Process management.** Ensuring organizational processes (e.g., testing) are followed helps streamline the lifecycle.

- **Teamwork.** Controlling interactions between developers working on a product or project can make it easier for team members to work together.

To perform these functions, you need to understand how to incorporate practices into your team's day-to-day workflow and also provide mechanisms to support and automate SCM functions.

As stated earlier, the build process provides support for performing SCM functions. As such, a build process that includes continuous integration (CI) and automated testing does this by:[2]

- Enabling reproducibility across the lifecycle, from the developer workspace through the production build environment, supporting configuration control and identification
- Providing a mechanism for the execution of automated testing in a repeatable environment, thus supporting process management, audit, and review
- Providing a logical place to report on code metrics and to deploy standard artifacts to a shared repository, thus supporting status accounting and teamwork

When combined with practices that associate commits to the source code repository (with associated requirements), your build system provides a central location to report on feature progress, which is further in support of status accounting.[3]

BUILD, ORGANIZATION, AND ARCHITECTURE

Because SCM and the build are tools to facilitate communication and manage change, they work in the context of your team so there isn't a single approach that will work with all organizations, all architectures, and all development approaches. To find the SCM approach that most effectively enhances communication and improves delivery speed, you need to consider the organization, the architecture, and the development approach.

According to Conway's Law,[4] the organization's structure establishes communication patterns and sometimes can affect the architecture. Consider the size and location of your team. A smaller, colocated team will need fewer formal processes to maintain working, coherent code. In fact, more process might disrupt effective communication patterns. A small team can be more successful working off a single code line, branching only for releases.[5] A larger team may benefit from a more structured approach, more automated processes to enforce policies, and staged builds to minimize the cost of broken builds.

Architecture is a key aspect of communication in the product team.[6] It also can make it easier or more difficult to manage change. A more modular architecture can lead to autonomous work among even noncolocated groups. A monolithic architecture might require more structured build and SCM processes. Also, the build process provides a clear model of the component architecture, making dependencies explicit.

The development approach has an impact on communication and architecture. A large team using a waterfall-like approach to development might benefit from more code lines and a staged approach to integration. An agile team that develops "test first" might find such an approach frustrating and could deliver more effectively with a single code-line model. Also, agile teams developing test first will tend to have more modular architectures, making it easier to manage change at the component, rather than the code-line level.

Because each team is different, you want to establish practices that give you the core functions of SCM without disrupting the productivity of your team. You want to focus not simply on "identification," "control," and "accounting and review" as functions, but rather on how these functions help you deliver value. While not as simple as applying a process checklist to your team, you will end up with a more effective team if you step back and consider what processes achieve the functional goals of SCM with the least overhead.

BUILD AND SCM IN THE LIFECYCLE

To evaluate how well your SCM process works, consider each of the following scenarios:

- **Having a new developer join your organization.** An effective SCM process should make it easy to have a new developer ready to check code within hours of starting a project.

- **Fixing an urgent production issue.** A developer should easily be able to identify which code line to make the change on and should be able to quickly create a workspace to enable implementation and testing a fix. The QA team should be able to test a new build shortly after a fix is checked in.
- **Gathering metrics on code quality and keeping the code working.** You should be able to quickly identify when tests fail or when test coverage goes below your desired threshold by looking at a report that your CI system generates.
- **Evaluating the latest version of the application.** If you can easily get access to a build that you can deploy and run, you will be able to quickly demo a new feature to a client or deploy an application for QA testing.
- **Identifying the code that you are using to verify compliance with licensing rules.** Your build system can generate reports and flag potential issues.

All these functions are supported by your build and SCM processes. The combination of your build and SCM mechanisms allows you to easily create working copies of your code. And a build and CI mechanism provides you with a dashboard on the health of the project.

AVOIDING COMMON MISTAKES

The most common mistakes teams make when considering implementing practices that will support the core SCM functions arise out of the belief that SCM functions must be heavyweight to be valid. This leads to either (1) ineffective (or nonexisting) processes, out of fear that "process" will slow the team down excessively or (2) overly involved processes, out of a fear that the processes (not the results) need to be highly visible to be effective.

It's possible to create SCM processes that add value while not being overly invasive or disruptive and which support how developers work day to day.[7] If developers have a process that supports their work, rather than being a "necessary evil," the process will be more effective, and you will need to invest less energy in enforcement of "overhead" activities. Automate the routine and provide tooling to reduce the cognitive overhead of processes. To find the right balance:

- **Prioritize making day-to-day tasks simple and direct.** Making it easy to check out and build code will not only impact daily productivity but will also enable more robust configurations. Developers have a role to play in helping with issue identification.[8]
- **Don't focus too much on tools.** Start with a process that makes sense and find a tool that supports it.
- **Integrate issue-tracking and SCM systems in a way that fits developer workflows.**[9]

Another common mistake is to confuse good SCM with "good branch management." While branching can be a useful tool to manage change and variation, it's not the only tool your team has. Any time you create a branch, you create another stream of code that has the potential to distract the team from the main line effort. Sometimes the benefits of the branch outweigh the costs, such as maintaining a release line to provide support for the current version while reducing the impact of bug fixing on new development. In many other cases, having more branches can distract from efforts to make your process able to deliver new releases frequently enough and of high-enough quality.

BETTER SCM, QUICKLY

Each organization's effective SCM process might look slightly different, but there are some common practices and tools that are essential. Setting up a CI system for your project that will build and run tests periodically will encourage the team to create a build that is maintainable and enable you to quickly identify issues that will make the development, testing, and deployment of your application difficult and error-prone. At a minimum, you should start with the following steps:

1. Place all artifacts necessary to build the code under version management.
2. Create a portable build.
3. Create a CI system.
4. Think about deployment at the start.[10]

Once you have these steps in place, you can find the policies and processes around change management, issue management, and branching that help your team develop code effectively while keeping stakeholders informed of status. And you can identify roadblocks to agility.

CONCLUSION

Having good SCM and build processes can help people be productive sooner, whether they are joining an organization or joining a team. While the right approach for your team will depend on your organization, architecture, and processes, there are some concrete steps you can take to bootstrap your way to a more effective SCM process that not only provides for accountability and traceability but also helps deliver value.

ENDNOTES

[1]Dart, Susan. "The Past, Present, and Future of Configuration Management." Technical Report CMU/SEI-92-TR-8, SEI, Carnegie Mellon University, July 1992 (www.sei.cmu.edu/reports/92tr008.pdf).

[2]Duvall, Paul M., Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007.

[3]Berczuk, Steve. "What Are You Doing?" StickyMinds.com, 14 March 2011 (www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=16706&tth=DYN&tt=siteemail&iDyn=2).

[4]Conway, Melvin E. "How Do Committees Invent?" *Datamation*, April 1968 (www.melconway.com/research/committees.html).

[5]Berczuk, Stephen P., and Brad Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley Professional, 2003.

[6]Grinter, Rebecca E. "Using a Configuration Management Tool to Coordinate Software Development." *Proceedings of the Conference on Organizational Computing Systems (COCS)*, ACM, 1995; Coplein, James O., and Gertrud Bjørnvig. *Lean Architecture: For Agile Software Development*. Wiley, 2010.

[7]Berczuk and Appleton. See 5.

[8]Bercuzuk. See 3.

[9]For suggestions, see 3.

[10]Berczuk, Steve. "Devops: Beginning with the End in Mind." Cutter Consortium Agile Product & Project Management *Executive Update*, Vol. 12, No. 6, 2011.

ABOUT THE AUTHOR

Steve Berczuk is an engineer and ScrumMaster at Humedica, where he's helping to build next-generation clinical informatics applications based on SaaS. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Mr. Berczuk is passionate about helping teams work effectively to produce quality software. He has a master's degree in operations research from Stanford University, a bachelor's degree in electrical engineering from MIT, and is a Certified Practicing ScrumMaster. He can be reached at steve@berczuk.com.