Crossroads News
A Monthly Publication for Software and CM Professionals

Agile Configuration Management Environments

Brad Appleton – April 2003

How can software configuration management be compatible with agile software development? Aren't the two diametrically opposed to one another? Sometimes it may seem that way. There is a commonly perceived tension between SCM and development agility that makes it difficult to achieve an effective yet precarious "equilibrium" between the two:

> Often people think of configuration management and version control as process-heavy things that might get in the way of the "real work" of coding. For many projects, SCM does get in the way, and some organizations overcompensate and don't use the tools to help them because of a fear that a process is inherently limiting. Other organizations want control and have so much process around version control that they hurt themselves. The right amount of version control is appropriate in agile projects...
>
> Often conflicts about software configuration management arise because of the difficulty of determining how much structure you need. Too little and chaos reigns, too much and the environment becomes stagnant...Highsmith also observes that "one of the reasons for this divide between process and practice is often the perception that an onerous process reduces the incentive to use any process." This reflects the reality that what matters is not your process as much as what people actually do. *[1]*

What is Agility?

Cockburn and Highsmith *[2]* define *Agility* as "the ability to both create and respond to change in order to profit in a turbulent business environment." They follow-up by saying "What is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability."

Agile development methods arose from a climate in which the business and technology worlds move at high speeds with high uncertainty and rapidly changing customer needs and priorities to survive in a turbulent and competitive market. The only way to survive in this environment is to become lean and nimble, capable of rapid response to change and churn.

Achieving Agility

Achieving this capability while avoiding unproductive thrashing of the project requires high quality communication and tight feedback loops between competent team members and with project stakeholders. This allows the project team to iteratively grow the product architecture and functionality despite ever-changing customer requirements and priorities.

Responding quickly and effectively to change is easiest to do when you can minimize the following *[3]*:

- the cost of knowledge-transfer between individuals
- the amount of knowledge that must be captured in intermediate artifacts
- the duration of time between making a project decision, and exploring its results to learn the consequences of implementing that decision

So achieving agility hinges upon reducing the time and cost of human learning *[4]* and its associated human activities of knowledge exploration, acquisition, transfer, and capture (and, ultimately, codification) *[5]*. This makes close collaboration and frequent iteration critical to the success of agile development.

Agile Development Characteristics

For this reason, agile methods focus upon the strengths of people and teams more than they do upon rigorous processes or detailed documentation. Processes don't develop software. People do! Therefore the agile process must be molded and shaped to make the process serve its practitioners (not vice-versa) *[6]*.

This gives rise to an agile "world view" based upon the set of agile values and principles put forth in the *Agile Manifesto [7]*. Projects employing the methods founded upon these principles and values share the following key characteristics:

- **Adaptive** – plans, designs, and processes are regularly tuned and adjusted to adapt to changing needs and requirements (as opposed to predictive methods that attempt to develop comprehensive and detailed plans/designs/requirements "up front") *[6][7][8]*
- **Goal-driven** – focus on producing end-results (working functionality) in order of highest business value/priority *[7]*. "Agile approaches plan features, not tasks, as their first priority because features are what customers understand" *[9]*. (Contrast with being task-driven, document-driven, process-driven, or risk-driven)
- **Iterative** – short development cycles, frequent releases, regular feedback *[1][7][8]*
- **Lean** – simple design, streamlined processes, elimination of redundant information, and barely sufficient documentation and methodology *[6][7][9]*
- **Emergent behavior** – quality systems (requirements, architecture, and design) emerge from highly collaborative, self-organizing teams in close interaction with stakeholders *[6][7][9]*

SCM versus Development *(Great Walls of Ire)*

Various IEEE standards partition configuration management into at least four areas (with a fifth area recently acknowledged in *[10]*):

- Configuration Identification
- Configuration/Change Control
- Configuration Status Accounting and Reporting
- Configuration Audit and Review
- Build and Release Management

Many of these aspects of SCM have long been considered by developers to be overly formal, rigid, and bureaucratic. At the same time, many software configuration managers and SQA professionals often perceive developers as undisciplined or ignorant of SCM concerns, constantly compromising product quality or integrity in the name of schedule or development speed.

The truth is that both sides are both right and wrong! The real problem is that they are on two different sides instead of on the same side. Many organizations segregate development responsibilities and SCM responsibilities almost exclusively to disparate people and teams in isolated activity-sequences of the development process lifecycle: development essentially delivers to SCM by "throwing it over the wall" to the next group down-stream in the lifecycle for integration, build, and test.

This "wall" all too often forges a barrier to effective communication and collaboration between developers and SCM people: each of these two "cultures" becomes indifferent or resentful to the concerns of the other and the impact upon themselves and each other. The resulting organizational dysfunction creates a self-sustaining "vicious cycle" that grows increasingly more problematic and adversarial over time. Something must be done to break the cycle and bring both sides together!

Agile Software Configuration Management (Agile SCM)

Effective software configuration management is crucial to the success of agile projects *[11]*. The key to understanding "agile" SCM is recognizing that in order to succeed in an agile environment without breaking agility, SCM processes and practices must embody the agile values and principles to achieve the characteristics of agile development.

SCM is a "whole team" responsibility
First and foremost, SCM people and developers must work in close collaboration toward the shared goal of successfully meeting a project's business and technical needs and requirements. Ideally, they should all be part of the same team. SCM should be a shared responsibility that is part of every team member's day-to-day tasks and activities:

- Everyone is responsible for regularly integrating, building, and testing in their own private workspace *[1]* before committing their changes to the repository.
- People who know how to build the system and design the build scripts collaborate daily with the people who design the architecture of the system. Often they are the same set of people and/or rotate the responsibility across team members. Everyone understands and appreciates the needs of both development and SCM because they experience the needs and benefits of both every day.
- If the build breaks, the whole team takes ownership of getting it working again (usually starting with the person whose changes caused the "breakage").

Responding to change versus controlling it
An agile project's tolerance for change must accommodate the rate of change of the specific business environment as opposed to some preconceived notion of how much change is acceptable. This is because agile projects are driven by business value rather than by conformance to the plan: "if we accept the notion of constant change and turbulence, then plans are still useful as guides, but not as control mechanisms…because they tend to punish correct actions" *[2]*.

Control is established by managing expectations with close communication and simple boundaries – "time boxes" in the form of iterations. After each iteration, expectations are (re)set and priorities are adjusted for the next iteration.

So configuration and change control procedures need to embrace rather than prevent change. Perhaps change control board (CCB) meetings should even be called "change planning meetings" to avoid the stigma of trying to control change rather than manage it.

- Such meetings occur at the beginning of each iteration to identify its feature content according to customers' latest priorities.
- The meetings need to be short and effective and include the business customer in the collaborative decision making process.

Authorization for developers to make changes needs to be instantaneous upon approval:

- Once a developer has signed up for a task, they should not have to wait to begin checking items out of the version control repository.
- And when the development team finds a bug that breaks the build or a fails a required test, they must be able to effect the repair without having to wait any noticeable period of time to receive "authorization".

Lean identification and reporting
Configuration identification is the set of all artifacts that describe one or more identifying characteristics of the end product and how to (re)produce it. *Configuration audits and reviews* become much more

streamlined and efficient when the set of configuration identification artifacts is kept to the bare minimum to suit the requirements of the project.

- Any required traceability tries to eliminate redundancy and rework by using fewer artifacts: Detailed requirements or use-cases may serve double-duty as acceptance test-cases; High-level requirements may exist in the form of feature/change requests and/or release notes (or be automatically generated using a template); Some end-user documents may even be used as use-cases or functional requirements documentation.
- Reports (and status accounting) for management and customers are also streamlined to the minimum required set, and communicated regularly and visibly. The reports should be generated quickly and easily, and preferably automated by simple utilities that can search, sort, and format integration/build/test results or integration/release plans and their requests/tasks.

Coordination and automation

- SCM tools and practices/processes cannot afford to hinder the agile development team or they won't be used at all. Tools and processes need to be simple, pragmatic, and enhance communication and coordination or reduce rework. The value they add to the project must be clearly understood and appreciated. In particular, use of tracking systems and version control tools must be minimally invasive with the goal of being transparent. The agile team won't readily tolerate an interruption of "flow" that comes from waiting for tools to collect data or complete lengthy operations.
- Use of SCM patterns *[1]* such as *Private Workspace*, *Mainline*, *Codeline Policy*, *Active Development Line*, *Private System Build*, *Workspace Update*, *Task-level Commit*, *Integration Build*, and *Smoke Test* will help each developer seamlessly perform their development and SCM duties on a daily basis while keeping the codeline clean and safely integrated, and the team well coordinated with one another's development efforts.
- It turns out that integration and test are forms of communication and feedback between developers and their changes. Hence frequent and regular integration/test is critical for successful agile development! So is effective use of branching and merging:
    - Branching should be done sparingly, creating new codelines only when parallel maintenance and development is unavoidably required. When branches are created, a *Mainline* should be used to maintain a manageable branching structure.
- Integration, build, and test activities are automated to the greatest extent feasible. Build tools/scripts, code structure, and network resources must be leveraged appropriately to minimize build times. One of the single biggest "drags" on development feedback cycle-time is the "friction" that comes from prohibitive build-times, or long testing-cycles that force development to either freeze or branch the code-base for significant periods of time while waiting for integration/build/test activities to complete.

Putting it all together: Opening, Middle game, and End game

The bottom line is that, to promote and sustain agility, SCM processes and tools must eliminate bottlenecks in feedback cycles by:

- Easing up on rigid front-end controls,
- Coordinating and streamlining development changes and communication, and …
- Automating back-end integration/build/test activities, increasing their frequency, and incorporating them into daily development tasks

References

[1] **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**, by Stephen P. Berczuk and Brad Appleton; Addison-Wesley, November 2002.

[2] *Agility* from **Agile Software Development Ecosystems**, by Alistair Cockburn and James Highsmith; Addison-Wesley, March 2002

[3] *Agile Software Development: The People Factor*, by Jim Highsmith and Alistair Cockburn; IEEE Computer: November 2001 (Vol. 31, No. 11), pp. 131-133

[4] *Retiring Lifecycle Dinosaurs*, by Jim Highsmith; *Software Test and Quality Engineering Magazine* (STQE), July/August 2000 (Vol. 2, No. 4)

[5] *Software is not a Product*, by Phillip Armour; Column "*The Business of Software*" Communications of the ACM August 2000 (Vol. 43, No. 8)

[6] *What Is Agile Software Development?*, by Jim Highsmith; Crosstalk – the Journal of Software Defense Engineering (special issue on Agile Software Development), Vol. 15, No. 10, pp. 4-9

[7] *The Agile Manifesto*, also appearing in *Chapter 1: Agile Practices*, pp. 3-11 of **Agile Software Development: Principles, Patterns, and Practices**, by Robert C. Martin; Addison-Wesley, November 2002

[8] *The New Methodology*, by Martin Fowler; created July 2000, revised April 2003; also published in abridged form as *"Put your Process on a Diet"*, Software Development Magazine; December 2000 (Vol. 8, No. 12)

[9] *Agile Software Development: The Business of Innovation*, by Jim Highsmith and Alistair Cockburn; IEEE Computer: September 2001 (Vol. 31, No. 9), pp. 120-122

[10] **Software Engineering Body of Knowledge** (SWEBoK), *Chapter 7: Software Configuration Management*, pp. 7.1-7.17; Version 1.00, May 2001

[11] **Configuration Management Principles and Practices**, by Anne Mette Jonassen Hass, Addison-Wesley, December 2002, *Chapter 18, pp.207-224* and *Appendix C: Agile SCM*, pp. 343-348.

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*

Sidebar: Manifesto for Agile Software Development *[7]*

Agile Values

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and Interactions *over Processes and Tools*

Working Software *over Comprehensive Documentation*

Customer Collaboration *over Contract Negotiation*

Responding to Change *over Following a Plan*

That is, while there is value in the items on the right, we value the items on the left more.

Agile Principles

- *Our highest priority is to satisfy the customer through early and continuous delivery of working software*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale*
- *Business people and developers must work together daily throughout the project*
- *Build project around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*
- *Working software is the primary measure of progress*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely*
- *Continuous attention to technical excellence and good design enhances agility*
- *Simplicity – the art of maximizing the amount of work not done – is essential*
- *The best architectures, requirements, and designs emerge from self-organizing teams*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly*