CM Crossroads
the Configuration Management Community

The Future of Agile CM by Brad Appleton

by Brad Appleton, Robert Cowham and Steve Berczuk

Together with our fellow columnists we have been indulging in a mixture of wishful thinking and crystal ball gazing to consider what the future holds for Agile CM. To misquote Malvolio, "Some things are born great, some things achieve greatness, and some have greatness thrust upon them." Rather than try to make grand predictions for 5 or 10 years down the road, we're mostly limiting ourselves to the next 1-3 years (except where noted of course)

What follows is a mixture of these, where the greatness referred to in the quote is our feeling for the impact they will have on CM and in particular Agile CM! We would also like to encourage you to let us know which ones interest you and which we should address in future columns.

Crystallizing our "Crystal Ball" Gazing?

We think the biggest impacts upon Agile SCM practices and tools will be in the following areas:

- Version Control
- Build/Test Management
- Distributed/Collaborative Development
- Traceability/Tracking
- Enterprise-wide CM Administration and Deployment
- Model-Driven CM

We'll elaborate on each of these in the following sections along with our ideas of which we think are near-term versus longer-term impacts.

And a little "Buzzword Bingo"!

Be it a fast-fading fad, or legitimate leap forward, the buzzwords and the hype "du jour" make marketeers do their best to show how their products and processes are buzzword-compliant and upto-date with the latest "hot" and trendy terminology. Why not avoid the rush and start today! What kind of real or lasting impact will these have on CM? Hard to say, but we'll venture a guess while the reader "racks up" bingo chips hunting down the following buzzwords throughout this article:

- Agile
- Patterns
- Object-Oriented
- Component-Based
- Model-Driven
- Aspect-Oriented
- Service-Oriented
- Enterprise Architecture
- Enterprise Integration
- Rule-Based
- Autonomic Computing
- On-Demand Computing
- Policy-Based Computing
- Grid-Computing
- RFID

Trends toward "Agile" Version Control
Back when software design patterns first became trendy and popular, a common "truism" from those tired of all the patterns-hype was to define a design pattern as "something that was omitted from the programming language". In many respects, the same is true of SCM patterns; one could define an SCM pattern as:

> *Something that was omitted from the SCM tool*

Granted, many SCM patterns are already built-in to many of the better SCM tools. Yet many of them are absent in the most widespread lowest-common-denominator SCM tools. As patterns and their practices gain greater industry-wide recognition and acceptance, the tool-industry will respond by adding automated tool support and higher-level concepts that remove increasingly more of the drudgery and complexity from the hands of development teams.

One significant factor here is the initial production release and growing popularity of Subversion - the de-facto successor to the most widely used and most-popular open-source version control tool, CVS (see http://subversion.tigris.org). Every conceptual improvement and pattern implemented in Subversion that was not in CVS has the potential to

assist a massively large user-base in progressing from more primitive operations and their elements to higher-level CM concepts and abstractions.

Project-Oriented Branches (*Streams*) represent a branch of evolution of an entire project and all the files developed within that project's scope (see *Streamed Lines* section on project-oriented branching). The ability to think and visualize release and delivery management in terms of such streams provides great visual power and simplicity for users in managing multi-project, multi-variant, and multi-sited development. Many tools have this already; more will follow.

Component-Based Versions (*Baselines*) correspond to not merely a single revision of a single file, but to a complete, correct, and consistent configuration of an entire product or component. Previously one had to manually apply a *tag* or *label* to a specified set of file revisions. It will be increasingly more common to define a container or component as an (evolving) set of items, with automated support for component-wide versioning and baselining.

Composite Streaming & Baselining combines together both streams and baselines to "virtual baselines of baselines" that is simply a reference to other baselines, and also to create a kind of "virtual" codeline of codelines (or "stream of streams") where the latest configuration of a such composite virtual stream is really an aggregate of the latest configuration of each of its constituent streams. Each time a "child" stream promotes a new baseline, it dynamically updates the "current configuration" of the composite codeline to refer to the latest baselines of the child component baselines. See references on Component/Container-Based SCM [1][2][3]

Task/Activity-Based Development (TBD) is already popular in many high-end version control tools. Yet developers more accustomed to lowest-common-denominator tools are less familiar with this feature and approximate it (often manually) via Task-Level Commit (TLC) and Change-Sets or Change-Packages. We predict that use of TLC will evolve into full-fledged TBD [4] as integration with project-management and change tracking tools becomes more prevalent in agile development shops.

Continuous Update [5] will ride along the coattails of TBD (along with finer-grained change tasks) and more tools will boast either virtual-file-system capabilities or else provide both "push" and "pull" synchronization mechanisms for updating private workspaces.

Improved Refactoring Support for renaming and splitting/combining files and directories/folders, along with directory versioning will also become a more commonly supported feature. Also look for language/syntax-aware merging [6] to make its way into popular IDEs (e.g. Eclipse).

Trends toward "Agile" Build/Test Management

To many, build management is the "bread-and-butter" of CM. Look for trends and improvements in build management architecture for Agile projects such as the following:

Continuous Integration and will continue to gain popularity and tool vendors will start advertising how their product provides or integrates with an existing Continuous Integration tool or mechanism (see cmwiki's ContinuousIntegration page)

Continuous Integration "Architecture" - look for individual projects and teams to avail themselves of all sorts of technology and tools to improve the frequency, reliability/quality/stability, and performance of both full and incremental builds. Asynchronous integration+build+test may even call upon technologies such as grid-computing, autonomic-computing, and on-demand computing to leverage and load-balance servers for building and automated testing. Expect a combination of open-source and commercial tools to be utilized.

Enterprise Continuous Integration/Staging - look for multi-group coordination of multiple component-teams to enhance their "build/integration architectures" to include dynamic event-triggered assembling and re-building of systems and subsystems from recently built+tested component versions utilizing multiple staging/integration areas (possibly distributed or replicated)

Testing Framework Integration - the agile/XP focus on test-driven development will drive a demand for better integration between SCM tools and not just unit-test automation tools (e.g., JUnit) but also testing frameworks. IBM/Rational already has a jump on this with the Hyades framework within Eclipse).

Trends toward "Agile" Distributed/Collaborative Development

As mentioned by some of our fellow columnists, the growth of "offshoring" and other forms of distributed development is making this aspect of CM tools much more important. How can we manage remote teams working on the same source base?

Will distributed-patch-based version-control paradigms such as that of Bitkeeper and Arch, where each workspace is a first-class repository, become more in-demand and more prevalent? If so, then this would make the need for things like private-branches and task-branches disappear (because you wouldn't need to "branch-in-advance of doing a commit"). The alternative approach is the stream-based model used by ClearCase/UCM, Accu-Rev, Subversion, Perforce, SpectrumSCM, Quartet, OurayCM and others. See David Wheeler's essay on centralized -vs- decentralized SCM systems

The distributed model has obvious performance advantages. However, corporate clients often prefer to have their repository managed centrally, despite of the potential performance implications. It can be much more secure and easier to ensure backups are in place ? they don't like the thought of software "inventory" (work-in-progress) sitting on desktops or laptops and not well managed.

In an interesting take on this, Joel Spolsky notes [7]:

> What's kind of surprising is that it has turned out to be easier to rewire the entire world for high-bandwidth Internet than it is to create good replication architecture so you can work disconnected!

Regardless of which way the trend goes (or whether it becomes a melding of the two), we are likely to see increased usage of and focus upon the following aspects of Agile CM:

- **Private Branches/Streams and Private Versions** - as stream-based branching and change-packages become more wide-spread, Task-Branches will give way to Private Branches [5] once the branch-itself is no longer the change-packaging mechanism. Developers will typically use a single workspace and stream (or just the workspace if Private Versions [5] are already implemented, as they are in BitKeeper, GNU Arch, and svk) for working on one task at a time, and will commit their change-packages to the codeline in finer-grained tasks, then begin work on the next task in the same private workspace. This will become more widespread as more people switch from CVS to Subversion and its contemporaries.

- **Continuous Coordination** - as mentioned before, the combination of continuous integration, continuous staging, and continuous update will bring greater demand to raise the level of awareness of activities going on in multiple private workspaces (even across multiple-sites). See papers under the Palantir project for one example [8], as well as [9][10][11][12][13] for more.

- **Workflow and Promotion Hierarchy** - if SCM is "the plumbing of software development" (as described in cmwiki's CMMetaphorsAndAnalogies) then look for the push of collaborative development environments to try and force SCM systems to make the transition from "plumbing" into "wiring" and "network infrastructure". Look for
    - Improved promotion modeling and mechanisms to ease heretofore manually or hand-crafted branch-and-tag-based promotion schemes
    - More/better integration with IDEs and with agile-friendly Project Management & Test tools, and even Instant Messaging, Collaborative editing environments (e.g., Wiki-Webs), and Content Management Systems (CMS). Look for new and improved collaborative environments like SourceForge, CollabNet and CodeBeamer that support integration of commercial and/or open-source solutions while at the same time providing much of their own value-added functionality in the areas of tool-integration, more sophisticated access-control and security schema, workflow, and status reporting.
    - Convergence of CM tools and Content Management Systems (CMS) capabilities
    - With the above in place, then a few more years down the road we may see dynamic, event-based publish/subscribe mechanisms that can trigger alerts, notification, status-reports, workspace-updates, change-propagation, build/change completion and promotion, etc., all of which can be used to trigger external applications (possibly integrate) and custom/configurable workflows at individual and team-wide, and site-wide scope.

Trends toward "Agile" Traceability/Tracking

Many "agilists" regard formal traceability as unnecessary bureaucratic overhead that breeds blame and distrust (instead of accountability) and creates intermediate artifacts requiring significant maintenance, all of which gets in the way of them producing working software. As such, some have been known to "rage against the machine" when it comes to formal traceability. This attitude will not be able to survive in the new age of Sarbanes-Oxley (and its equivalents elsewhere, such as the new Companies Bill in the UK) and increasingly more frequent, more severe, and more mission-critical security threats.

As we forge ahead into the world of wireless connectivity and seamless mobility where everything is connected, with self-aware service discovery and all it portends for the security not just of our data, but the applications that increasingly run more and more of the day-to-day infrastructure that we take for granted like the air we breathe, security and accountability (and hence traceability) will be an ever-increasing, omni-present legal and professional obligation in all aspects of software development.

This will become an even greater concern with even scarier implications with the coming viability of nanotechnology, wearable computing, and bio-technology for medical/health devices that monitor, regulate, or even diagnose human functions.

With all of this, there is no stopping the traceability juggernaut, even in areas of program where it previously may not have been mandated by an organization, the industry itself will do so. The key factors are the requirement for audit and the fact that directors and other responsible people can be imprisoned for signing a statement that later turns out to be false, particularly if there are catastrophic financial and/or life-threatening results.

It is as yet unclear what the implications of this are and how the audits will work in practice. What is certain is that the threat of imprisonment is concentrating the minds of a lot of people who wish "appropriate procedures" to be put in place.

While this will obviously affects the whole of CM, it is going to be particularly interesting to review the effects on agile initiatives. In a previous column [14] we reviewed agile methods, and the concern about them from the point of view of traceability. For example, how does a director know that there are no backdoors in the products produced by his or her company? Does it make a difference if the development team used XP and the requirements are only stored on index cards?!

This is going to be a huge area. Indeed, recently, there has been a rash of seminars announced on this subject as consultancies jockey for the privilege of educating worried directors and holding their hands through implementation and deployment.

The impact upon effective yet necessary CM for agile development will be very large indeed, and many are already starting to see and feel it. In CM terms we need to quickly and easily see the status of any change, version, request or build provided:

- **Task-Based Development** will play an important role here too**:** Some kind of task-based scheme (see Austin Hastings article [4] on how Task-level commit (TLC) is one ingredient on the road to task-based development (TBD)) will be a key plug-point for integrating with (and tracing to) project management activities, estimates, requirements, tests, and verification/validation/auditing.
- **Real Tracking Tools** - the fervor to resist automated project & change tracking tools in favor of index card will have to give way to the inevitable realities of mandated traceability. Cards may still be used as an effective means of engaging in dialogue and a tactile mechanism for gathering and enacting stories and scenarios. But even then they will need to be supplemented with automated electronic tools that connect a projects "tracking" database with that of the rest of the organization in order to perform effective portfolio management at the organizational level, as well as search, sort, query, report, and "refactor" issues across all projects or products in an overall program.
- **Dynamic Event-Based Traceability** (such as that attempted by the SABRE project and DePaul University Requirements Engineering lab) will become more desirable as a means of automating much of the requirements traceability burden. Look for hand-crafted solutions to be scripted together initially (some might even make use of the new annotation and metadata features in J2SE 5.0), and then for commercial tool-support and/or open-source efforts a few years afterward.
- **SCM and Requirements Repository Integration/Unification** - typically, source code version repositories are separate from the repositories used by requirements management tools such as DOORS, Requisite-Pro, and CaliberRM. This poses a significant enterprise integration and deployment/upgrade/support burden on organizations trying to integrate the two in order to facilitate traceability and change-tracking:
    - The Requirements Management tools typically have wonderful facilities for attaching and viewing meta-data and links and taking a container-based view of collections of items, while lacking considerably in areas of branching and merging and baselining when it comes to having to support multiple projects/variants/sites. The Source Code version control tools have the opposite problem.

- Look for vendors to provide a better out-of-the-box integration to this problem in the near-term, For the mid-term, look for vendors to start storing both requirements and code in an integrated or unified repository so that both can enjoy the benefits of state of the art versioning capabilities along with state-of-the-art metadata/linking and container-based editing capabilities. MKS Requirements already has a jump start on this with their recent requirements management offering (already billed as "lightweight/agile"). Look for fine-grained versioning [15] to play a role in this as well.

- **Lean Traceability** - the last bastion of resistance against any traceability will be attempts at *Lean* traceability:
  - *Modularity & Granularity*: Efforts to eliminate redundancy and complexity will result in using smaller grained artifacts and having them all "connected" or linked together in a database and/or perhaps an XML Schema using SOAP and other enterprise integration technologies.
  - *Auto-Generation and Locality of Reference*: There will be renewed attempts to co-locate related information in the same object rather than having to split and track separate objects. And some kind of "literate" document/traceability generation ("build") will extract the necessary information and present/report for the proper stakeholders. This will plug-in to IDEs such as Eclipse (see [16]).
  - *Encapsulation and Architecture*: Traceability will be done to the largest-grained object scope possible, and intra-object dependencies will be aggressively refactored to remain as cohesive as possible while maintaining low-coupling with other objects (of all types, not just code)
  - *Aspect-Oriented Traceability*: Ivar Jacobsen [23] writes that use-cases are themselves logical entities that cut-across an architecture, and as such are a natural fit for applications of aspect-oriented development. (Look for AOP and related tool-support to be applied in this manner *before* it is applied to process architecture.)

- **CM RFID?** On a related note, it may be only a matter of time before l things like RFID have their equivalent in CM with regard to traceability:
  - The brother of one of the authors was a logistics officer with the British army in the second Gulf War. He noted the difference an active satellite based RFID based tracking system made in tracking certain types of shipment (only high value items). The financial cost of duplicates being ordered and stores going to the wrong place is massive, let alone the consequences of vital stores not being delivered to front line troops in desperate need.
  - Unfortunately, for the majority of stores with passive RFID and imperfect systems in place, things still could be lost once they had left ships. The army recognizes that they need to beef up the system and automate as much as possible. The more people are required to take manual steps, even if that's to pass a reader near a passive RFID tag, the greater the chance of mishap.

Trends toward Enterprise CM Administration/Deployment

Life will start to get better for us poor souls who have to administer, deploy and support full-fledged SCM solutions across an entire enterprise (agile or otherwise). Things may actually get worse before they get better, because it may take some time to get different technologies and standards sorted out before the winners begin to emerge. Expect to see things like the following:

Full Application Lifecycle Management - look for the scope of CM (even enterprise CM) to be expanded to include activities even earlier in the product lifecycle, such as product management and marketing, portfolio management and planning, integrated systems engineering; as well as later in the product lifecycle to deployment/install/upgrade, and even run-time configuration management & monitoring of run-time components and web-based services. Expect vendors of integrated tool-suites (IBM/Rational, Telelogic, Borland, etc.) to expand their offerings of integrated solutions accordingly.

Enterprise CM Integration Architecture - IT architectures and enterprise application integration of CM tools with tools for requirements management, project management, test-management, and their corresponding repositories and reporting mechanisms will abound.

Enterprise Data Management - Enterprise data warehouses ("data marts") will become increasingly more common as a means to both integrate the business-intelligence and decision-support data from multiple disparate repositories and tools while at the same time offloading some of the real-time transactional processing from the native tool/repository to the data-mart.

Web-enabled "CM Services" and Service-Oriented CM Architecture may be utilized to provide "on-demand" CM services [17] for SCM auditing/reporting, status accounting, and traceability via a single web portal serving as an SCM "information radiator" to all subscribed parties and projects. They will also help to ease the customization and deployment of CM solutions for overall decreased "Total Cost of Ownership"

Fast-forward to Model-Driven CM "Factories"

Model-Driven CM will be longer in coming (5-10 years, probably 2010 at the earliest). As Model-Driven Architecture/Development (MDA/MDD) gain more traction, the progression toward "**Software Factories**" (a perhaps misleading name for a very powerful vision from Microsoft [21]) will become more of a reality. When it does, look for the lines to blur between the modeling environment and the development environment:

Conceptual (rather than Physical) CM entities - Logical/conceptual entities in the UML model (e.g., components, packages, classes, interfaces and methods) will become the primary means of visually navigating through the code, rather than the current predominant use of files and folders/directories. Before long, only CM folks will have to have more than a passing familiarity with how the systems maps into files and folders for optimal build cycle-time and dependency management, and they will be able to use the model and its modeling facilities to track and record dependencies and traceability. So look for Configuration System modeling to become popular once again [22] and finally make its way from research into industry.

Adaptive automated build configuration and optimization - We may then be able let the tools worry about how to perform the translation into files and folders in order to optimize build+test cycle time, and have configuration settings and "profiles" for CM/Builders to tune and tweak to automate computation, display, and tracking of build-dependencies, minimal rebuild/recompilation/relinking, and make a single consistent turnkey mechanism for doing private builds, integration builds, and release builds. This relates to recent build products for distributed builds and build server, ranging from the use of freeware such as Cruise Control, Draco and AntHill, to more commercial tools such as BuildForge, OpenMake, AntHill Pro, and Electric Cloud.

Fine-grained versioning - when conceptual entities become the means of navigating through the code, these finer-grained entities will also become the units of checkout and checkin (see the Stellation subproject of Eclipse [15]). This will decrease the likelihood of checkout contention and merge conflicts, but will increase the need for the greater dependency tracking and analysis that MDA/MDD will enable. This will also demand even tighter integration between the modeling tool and the IDE with each other, as well as with the SCM tools.

Inter-repository integration/unification - currently, different tools and repositories tend to be used for requirements management, design models, and source-code version control. In the short-term, better integration between these different version control repositories will be necessary in order to perform effective change and configuration control. In the longer term, there will be a clamor to be able to use a single object-based repository and integrated database to version all types of artifacts across the entire application lifecycle.

Aspect-Oriented CM - if requirements (use-cases), design (models), and coding artifacts all end-up in the same repository, or at least in the same integrated CM system, there will be a desire to apply basic common policies across the board with some activity/artifact-specific tailoring.

- "Aspects" are crosscutting concerns of a system (e.g. examples may be logging and security). And since CM is a cross-cutting, integral part of a project's architecture, communication, and organization, SCM policies and practices could be automated with aspect-oriented technologies.

- AOP concepts and technologies [24] allow a single set of concerns/policies to be applied across all parts of the lifecycle with activity/artifact-specific tailoring ("advice") to be customer-configured and applied at the points of policy instantiation ("pointcuts"). This would facilitate policy-based and rule-based CM practices and business/workflow process enactment [25][26][27]

Conclusion

This has just been a brief taste of some of the issues. Hopefully it has provided food for thought, and as mentioned at the start, let us know what you would like to see covered in future columns.

References and Additional Resources

[1] *Container-Based SCM and Inter-File Branching*; by Laura Wingerd; BSC CMSG 2002; (http://www.bcs-cmsg.org.uk/conference/2003/papers/wingerd.pdf)

[2] *Introduction to Container Based Software Configuration Management*; by Hans Thelosen, March 2002; (http://www.dse.nl/~thelosen/artikelen/introduction_to_scm.pdf)

[3] *Flexible Component-based SCM*; by Tom Brett; BSC CMSG 2004; (http://www.bcs-cmsg.org.uk/events/e20040317/Brett%20Flexible%20Configuration%20Management.pdf)

[4] *SCM Patterns: Building on 'Task-Level Commit'*; by Austin Hastings; CM Crossroads Journal, June 2004 (http://www.cmcrossroads.com/article/28269)

[5] *Codeline Merging and Locking: Continuous Update and Two-Phased Commits*; by Brad Appleton et.al, CM Crossroads Journal, November 2003 (http://www.cmcrossroads.com/articles/agilenov03.pdf)

[6] *A State-of-the-Art Survey on Software Merging*; Tom Mens; (http://scgwiki.iam.unibe.ch:8080/Release/uploads/8/MergingSurvey.pdf)

[7] *The Shlemiel way of software ? Interview with Joel Spolsky for Salon Magazin*e, (http://www.salon.com/tech/feature/2004/12/09/spolsky/)

[8] *Continuous Coordination: A New Paradigm for Collaborative Software Engineering Tools*; by Andre van der Hoek et.al; available from the "Palantir" project at http://www.ics.uci.edu/~asarma/Palantir/publications.shtml (PPT at http://www.erenkrantz.com/Geeks/Research/presentations/Palantir%20Presentation.ppt)

[9] *Building Collaboration into IDEs* [be sure to check out the RESOURCES section of this one] (http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=104&page=1)

[10] *Breaking The Code: Moving Between Private & Public Work In Collaborative Software Development*; by David Redmiles et.al; ICSGW 2003 (http://www.ics.uci.edu/~redmiles/publications/)

[11] *Tactical Approaches for Alleviating Distance in Global Software Development*; by Erran Carmel and Ritu Agarwal; IEEE Software, March/April 2001 (http://www.american.edu/ksb/mogit/carmel/ieeesw2001.pdf)

[12] *Unifying Artifacts & Activities visually for distributed software development teams*; Jon Froehlich et.al.; ICSE 2004; (http://drzaius.ics.uci.edu/~jfroehli/augur/pubs/froehlich_j-ICSE2004.pdf)

[13] *A Weakly Constrained Approach To Software Change Coordination*; by Ciaran O'Reilly; ICSE 2004; (http://delivery.acm.org/10.1145/1000000/999409/21630066.pdf?key1=999409&key2=5364055011&coll=GUIDE&dl=GUIDE&CFID=35807002&CFTOKEN=65347143)

[14] *The Agile Difference for SCM*; CM Crossroads Journal Oct 2004, (http://www.cmcrossroads.com/article/34375)

[15] "Stellation" subproject of Eclipse http://www.eclipse.org/stellation

[16] *Chianti: A tool for change impact analysis of Java Programs*; by Xiaoxia Ren et.al. OOPSLA 2004; (http://www.prolangs.rutgers.edu/refs/docs/oopsla04.pdf)

[17] *The Past, Present, and Future of Configuration Management*; Susan A. Dart (ftp://ftp.sei.cmu.edu/pub/case-env/config_mgt/papers/PastPresentFuture.pdf)

[18] *Release Management ? Making it Lean and Agile*; by Robert Cowham et.al., CM Crossroads Journal, November 2003 (http://www.cmcrossroads.com/article/31243)

[19] *Continuous Staging: Scaling Continuous Integration to Multiple Component Teams*; by Brad Appleton et.al; CM Crossroads Journal, August 2004 (http://www.cmcrossroads.com/newsletter/articles/agilemar04.pdf)

[20] *Streamed Lines: Branching Patterns for Parallel Software Development*; by Brad Appleton et.al.; 1998 Pattern Languages of Program Design; (

http://acme.bradapp.net/branching/)

[21] Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools ; Jack Greenfield et.al; John Wiley & Sons, 2004**.** See related articles:

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/softfact3.asp
- http://theserverside.net/articles/showarticle.tss?id=SoftwareFactories
- http://allconsuming.net/item.cgi?isbn=0471202843
- http://www.sei.cmu.edu/SPLC2004/ (See keynote address by Jack Greenfield)

[22] *System Modeling Resurrected*, Andre van der Hoek, et.al. (http://www.ics.uci.edu/~andre/research/papers/SCM8.ps)

[23] Aspect-Oriented Software Development with Use Cases ; Ivar Jacobson, Pan-Wei Ng; Addison-Wesley, 2005 (see http://www.awprofessional.com/content/images/0321268881/samplechapter/jacobson_ch11.pdf)

[24] Aspect-Oriented Software Development ; Robert Filman et.al.; Addison-Wesley, 2005 (see http://www.awprofessional.com/articles/printerfriendly.asp?p=340868)

[25] *Dissecting CM Policies*, Andre van der Hoek, et.al. (http://www.ics.uci.edu/~andre/research/papers/SCM11-1.pdf)

[26] *A Testbed for Configuration Management Policy Programming*, Andre van der Hoek, et.al. (http://www.ics.uci.edu/~andre/research/papers/TSE2001.pdf)

[27] *A Generic, Reusable Repository for Configuration Management Policy Programming* , Andre van der Hoek, et.al. (http://www.ics.uci.edu/~andre/research/papers/CU-CS-864-98.ps)

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.

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

Robert Cowham is the founder of Vaccaperna Systems providing SCM consultancy and training to organisations. With 20 years of experience in software development, he has long had an interest in SCM, and has worked with clients around the world in this arena during the last 7 years. He is on the committee of the CM Specialist Group of the British Computer Society for whom he has organised several events, including their 2-day conference in 2003 (and their upcoming 2005 event). He has a BSc in Computer Science from Edinburgh University and is a Chartered Engineer (CEng MBCS CITP). You can contact him at rc@vaccaperna.co.uk