Software Configuration Management: State of the Art, State of the Practice

Karol Frühauf[1] and Andreas Zeller[2]

[1] *INFOGEM AG*
Informatiker Gemeinschaft für Unternehmensberatung
Rütistrasse 9, CH-5401 Baden, Switzerland
Karol_Fruehauf@compuserve.com
[2] Universität Passau
Lehrstuhl für Softwaresysteme
Innstraße 33, D-94032 Passau, Germany
zeller@acm.org

Abstract. Which are the open problems in Software Configuration Management (SCM)? The purpose of this paper is to ignite a discussion on current and future SCM directions. Based on the findings of a Dagstuhl Seminar on the current state of Software Engineering, we *assess the state of SCM* with the goal to identify effective SCM tasks and solutions, to establish a core body of SCM knowledge, and to denote remaining real-world SCM problems.

1 Introduction: An Assessment of SCM

Which are the open problems in SCM? Software Configuration Management (SCM) is one of the few success stories in Software Engineering. All software organizations admit the importance of SCM as a prerequisite for a coordinated software development. Consequently, SCM is widely used—and with success. Already, the SCM tools market is expected to be worth over a billion dollars [4].

The 1999 Dagstuhl Seminar on "Software Engineering Research and Education: Seeking a new Agenda" [7] has joined experts in several Software Engineering fields to take stock of the current state of Software Engineering research and education. Within this Seminar, we have addressed this task for the SCM area—*assessing the state of Software Configuration Management*. In particular, we have attempted to cover the questions:

What do we know? Which are the *SCM tasks and solutions* that every practicing software engineer should be able to perform?
What should we teach? Which is the *core body of SCM knowledge* that has been validated as useful in practice?
What should we know? What are the most important *open SCM problems?*

In contrast to earlier approaches, we have not searched for novel ideas "that should keep researchers busy for the next several years" [31] or examined possible similarities between some area X and SCM [10, 34], but attempted to identify remaining *real-world SCM problems*—problems faced by today's practitioners, yet not sufficiently addressed by SCM research. This paper summarizes our results; its purpose is to ignite a discussion on current and future SCM directions.

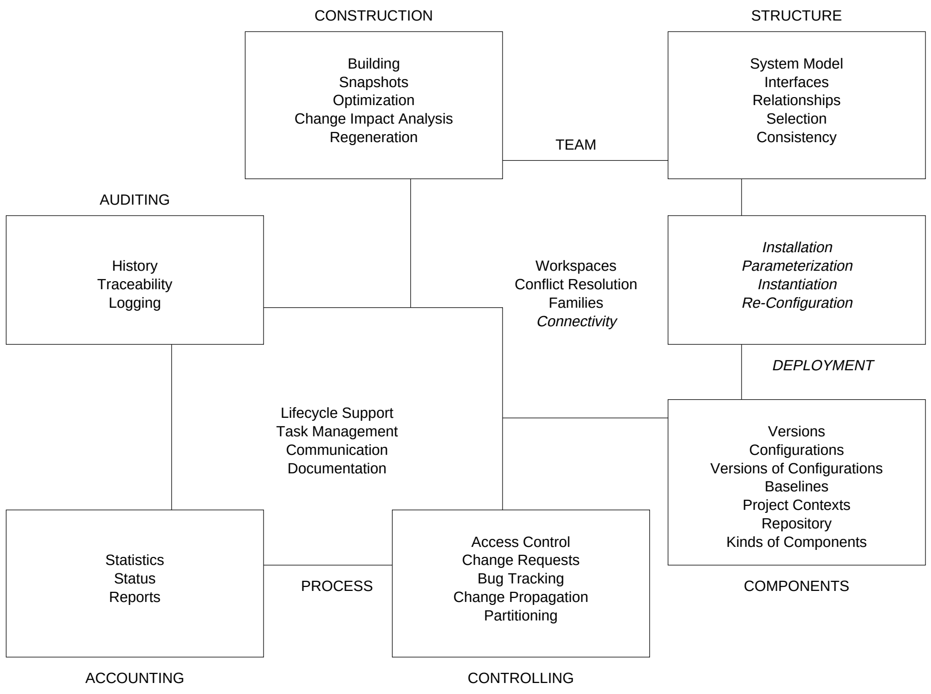

Fig. 1. SCM Functionality Areas (after Dart [6])

2 Assessing SCM Solutions

In this paper, we have focused on those SCM solutions that are provided or supported by some automated SCM tool or system, be it a research prototype or a full-fledged commercial system. (This is just a matter of economy; if some solution has only been proposed, but never realized, we will not regard it here.) Note that we do not consider SCM organisational matters. The reason is not their irrelevance; in fact we are convinced that SCM work procedures need to be defined in an organisation before any SCM tool can be selected (which will of course backfire on the work procedures). The difficulty is that as every organisation has its own flavour of work procedures, there is no such thing as a solution.

Relying on Dart's survey [6], the functionality of SCM systems can be grouped into two major *functionality areas*, as shown in Figure 1.[1] The *team-centered* functionality areas deal with the *technical aspects* of SCM:

Components. Identify, classify, store and access the components that form the product.
Structure. Represent the architecture of the product.
Construction. Support the construction of the product and its artifacts.
Team. Enable a project team to develop and maintain a family of products.

[1] Areas in *italic* are areas not covered in [6] that are now considered part of SCM.

Deployment. Support the remote installation and maintenance of the product.

In contrast to the team-centered areas, the *process-centered* functionality areas cover management issues:

Auditing. Keep an audit trail of the product and its process.
Accounting. Gather statistics about the product and its process.
Controlling. Control how and when changes are made.
Process. Support the management of how the product evolves.

For each of these areas, we shall discuss the state of the art, assessing available and proposed SCM *solutions*—means that solve specific SCM tasks. Following the *assessment categories* as elaborated at the Dagstuhl seminar [7], each SCM solution is ranked according to five categories:

Effectiveness. How well does the solution work? This considers factors such as how much of the task it covers and how good a solution it is to the problem posed by accomplishing the task. Ratings are
- *high* (the solution is very effective),
- *medium* (the solution is somewhat effective), and
- *low* (the solution is hardly effective at all).

Affordability. The extent to which a typical software development organization can afford to perform the solution. (Note that it may be that a solution is high cost, but that an organization cannot afford not to use it.) Ratings are
- *high* (the solution is very affordable),
- *medium* (the solution is somewhat affordable), and
- *low* (the solution requires relatively high investment).

Teachability. The extent to which the solution can be taught in a university, including the body of knowledge that must be conveyed to students and how well we understand how to convey that body of knowledge. Ratings are
- *high* (we know how to teach the solution very well),
- *medium* (we know how to teach the solution to some extent), and
- *low* (we do not really know how to teach the solution).

Use in Practice. The class of users who have adopted the solution:
- *laboratory users* (LU) – researchers developing prototypes and models,
- *innovators* (IN) – willing to use early prototypes of the solution,
- *early adopters* (EA) – willing to use advanced prototypes
- *early majority* (EM) – willing to be the first users of industrial-quality versions of the solution
- *late majority* (LM) – not willing to use the solution until there is considerable industrial experience with it.

Research Potential. The extent to which further research is supposed to increase effectiveness, affordability, teachability, or use in practice. Ratings are
- *high* (major breakthroughs can be expected),
- *medium* (substantial improvements are likely), and
- *low* (details may be improved).

3 SCM Team Tasks and Solutions

We begin with a discussion of the solutions available in the team functionality area; Table 1 on the next page summarizes our findings.

3.1 Team

The notion of a *workspace* that isolates a developer from other's work is crucial to SCM. Generally, workspaces should provide their own structure, states for the configuration items and configurations, and access rights for the different functions in the project [14]. The extent to which these requirements are met systems varies from file-based checkin/checkout mechanisms as in RCS [28] over virtual file systems as in CLEARCASE [18] or in *n*-DFS [13] to database-supported workspaces in ADELE [9].

However, since every SCM system provides means to generate, propagate, and apply changes, every SCM system allows to *simulate* workspaces—even if the "workspace" is but a developer's private directory or a branch in the version graph. This problem is thus considered solved.

With an uncontrolled propagation of changes, the chances for two or more people's changes interfering with each other are high; this leads to *conflicts* that must be resolved. This *merging* of changes is still manual work. Textual merging is considered too unsafe for many environments; the effectiveness of syntactic merging [33, 3] and semantic merging [16, 2] has not yet been validated. Any solutions that ease the pain of manual conflict resolution are likely to save valuable developer time; here is still work to do.

A group working together needs *connectivity* to propagate changes. Given a small group with good interconnection, a central repository suffices for all project sizes. Things get more difficult for multi-site, multi-organization software developments (so-called *virtual software corporations*). Here, local copies of shared resources must be replicated and cached; remote access must be designed such that cooperation is possible while avoiding total project disclosure. Although several commercial SCM tools such as CLEARCASE [18] offer support for wide-area connectivity, the area remains subject to further research.

3.2 Components

Managing the history of individual components is a well-understood SCM task. Tools like SCCS [25] and RCS [28] are being used for more than two decades now. Efficient means to store and retrieve huge amounts of versions in a *repository* are available and have been thoroughly validated [17]. *Identifying* and reconstructing a configuration by means of its components or changes applied to a baseline is a task easily solved with all available SCM tools. SCM at the component level may well be the SCM area that is best understood of all.

3.3 Structure

Versioning of structures, i.e. systems of related components, is still not completely solved. Let us start with the inventory of components—the *system model*. Most SCM

	Ranking of SCM Solutions				
SCM Tasks	Effectiveness	Affordability	Teachability	Practical Usage	Research Potential
Team					
– *Workspaces* (individuals, groups)	high	high	high	LM	low
– *Conflict resolution* (parallel work as the rule, automated merging)	low	low	med?	IN	high
– *Local area connectivity*	high	med	high	LM	low
– *Wide area connectivity* (remote access, replication, caching)	med	med	med	EA	med
Components					
– *Version management for components* (revisions, branches, checkout/checkin, identification)	high	high	high	LM	low
– *Repository* (storage issues, deltas)	high	high	high	LM	low
– *Configurations* (baselines, parts lists, identification)	high	high	high	LM	low
Structure					
– *System model* (interfaces, relationships)	med?	low	low	LU	med
– *Version management for structures* (renaming, reorganization, retiring of subsystems with whole history)	med	high	high	EA	med
– *Selection* (baselines plus change sets, generic configurations)	high	high	high	EM	low
– *Consistency* (compatible versions)	med?	low	low	LU	med
Construction					
– *Building* (snapshots, optimization, dependencies)	high	high	high	LM	low
– *Regeneration* (integrated with SCM)	high	med	med	IN	med
Deployment					
– *Replication* (on a medium)	high	high	high	LM	low
– *Installation* (in a consistent manner)	med	med	low	EM	high
– *Parameterization* (customizing)	med	med	low	IN	med
– *Instantiation* (running)	med	med	low	IN	med
– *Reconfiguration* (dynamically)	low	low	low	IN	high

Table 1. SCM Team Tasks

systems do not go beyond simple part lists; relationships and interfaces are barely supported, let alone versioned. (Exception to this rule are *build dependencies*, as discussed in Section 3.4.) The extent to which system modeling is part of SCM is still being discussed [32].

Although several commercial SCM systems and even free tools like CVS [1] allow decent versioning of file hierarchies, issues like renaming or reorganizing structures are still not handled in a fully satisfying manner.

All SCM systems offer methods to *select* specific configurations; the range goes from tags as in RCS or CVS to elaborated rules as in CLEARCASE [18] or ADELE [9]. The organization of versions (or changes) within an SCM system, the *version model*, has considerable impact on the way users interact with an SCM system [5]. Although it has been shown that all existing versioning models can be unified to applying constrained changes [37], the quest for user-friendly and intuitive SCM interaction continues.

A still open problem is how to identify and denote consistent configurations in presence of multiple variants. So far, systems like ICE [36] or CMA [24] are confined to lab use only. On the other hand, one must ask whether variability at construction time—that is, permanent variants—is still an SCM issue. In general, product variability is best handled *by the product and within the product*. The more we design for change, the more we abstract from system issues, the less variability we have at construction time, and the more variability we have at run time. Although consistency issues may rise again within the scope of *dynamic reconfiguration* (see Section 3.5), efforts spent in variability may thus better be directed towards software design.

3.4 Construction

The SCM task of building products can be summarized as “MAKE rules”. Virtually every software product is built using MAKE [12] or one of its numerous descendants. Significant improvements on the original MAKE include smart recompilation [29], parallel and distributed building [27], automatic dependency tracking [18], or caching of derived versions [18]. All of these are widely used today, and it is difficult to see room for further improvements.

A more important problem is the traditional distinction between construction tools (i.e. MAKE) and SCM tools (i.e. RCS), as this separation hampers the *regeneration* of derived files. A notable exception and an example of good integration is the build facility integrated in CLEARCASE [18].

3.5 Deployment

Deployment is a new field of SCM, traditionally subsumed under “maintenance”. According to Heimbigner and Wolf [15], deployment encompasses *installation, parameterization, instantiation,* and *reconfiguration;* ISO 9000-3 also lists *replication.*

Replication means to make sure that the intended configuration is correctly and completely copied on the medium chosen for delivery. Copying from master to an EPROM, preparing a package with CD and paper documents, and putting the files to an area from where they are electronically transferred to the customer site are all techniques for replication. This is evidently a topic for process engineering in an organization. The main challenge is to define (and apply) it as a process with self-check so that the mistakes can be detected before delivery. We cannot see research opportunities in this area.

Installation is the task of transferring the product to the user. Basic installation is easy—a set of files is copied to places reachable by users. But this task becomes the more difficult the more the product depends on other products, maybe in specific versions. Managing these dependencies and denoting consistency is only partially solved today; the more applications depend on each other, the more the need for installation support will increase.

Parameterization is the task of adapting the product to the user's context—a task either done on site (by the user) or in the factory (especially when hardware is part of the delivery). Parameterization is traditionally carried out by customization files and environment variables; more recently. tools like GNU AUTOCONF [19] are used to determine system properties automatically. In future, such checks will be increasingly carried out at run time and will thus need system support; the Windows registry is a yet rudimentary form of such capabilities.

Instantiation is the task of starting the product into execution. This is trivial (and solved) for simple, monolithic applications, but becomes a challenge as soon as multiple components interact with each other; trading services like the ones specified for CORBA [22] can serve as base for determining a consistent configuration.

Reconfiguration means adapting the product to new requirements while it is executing. This includes all decisions made during installation, parameterization, and instantiation, and may also mean that the product entirely re-creates and replaces itself. This problem is well-understood when speaking of isolated applications: uploading software releases dynamically to space probes or telecommunication switches is common usage. However, dynamic reconfiguration will gain even more importance as more and more software products interact with each other for an undetermined time. The challenge for SCM is to see how far classical SCM concepts can be applied dynamically.

Our observation is that the aspects of deployment are considered late in the life cycle of a software product, usually after the software is finished and somebody discovers that it has to be shipped to the customer site. Very few requirements specifications contain paragraphs for requirements on deployment issues. Most software producers can improve in this area; complex software systems offer new research issues.

4 SCM Process Tasks and Solutions

Let us now turn to the process functionality area. Our findings are summarized in Table 2 on the following page.

4.1 Process

How far should the SCM system support the user's lifecycle model and their organization's policies? Although every SCM system comes with a built-in process in the small

	Ranking of SCM Solutions				
SCM Tasks	Effectiveness	Affordability	Teachability	Practical Usage	Research Potential
Process					
– *Lifecycle Support* (process enforcement)	low?	low	med	LU	med
– *Task management* (identify current and pending activities)	med	high	high	EM	low
– *Communication* (relevant events)	high	high	high	IN	low
Auditing (history, traceability, logging)					
– *of individual items*	high	high	high	LM	low
– *of structures* (related documents)	high?	med?	low	LU	med?
Accounting (status, statistics, reports)	high	high	high	LM	low
Controlling					
– *Access control* (no unwarranted changes)	high	high	high	EM	low
– *Change requests* (automatically)	med	high	high	EM	low
– *Bug tracking* (automatically)	high	high	med	IN	med

Table 2. SCM Process Tasks

(i.e. checkin/checkout-cycle, long transactions, etc.), the degree to which large-scale processes are supported varies.

Our experience tells us that the big leap forward is the clear *definition* of software processes. Use of tools is beneficial only if they are really supportive; often they take the role of bureaucrats increasing the number of required interactions for the developers. Consequently, SCM systems that are too rigid in enforcing a process will be cursed by developers and reduce effectiveness. The distinction between support and discipline and thus the effectiveness of lifecycle support remains to be validated.

Rather than *enforcing* activities, more advanced SCM systems offer means to *track* current and pending activities. *Task management* is an area overlapping with (project) management. If tools are used it must be carefully decided which type of information is kept in the SCM tool and which in a project management tool. The interface is thin if the SCM system handles the states of the configuration items (and configurations) and this information is used by the (project) management for the progress control. Tight coupling of work activities with the state control of the work results leads to sluggish SCM systems.

Ultimate process support is achieved with automated *work flow systems*. These are not widely used (yet); their validation is a pending research topic. In practice, work flow is typically organized by informal communication. Most SCM systems support *triggers* that are associated with specific events—such as automatic notification by e-mail whenever a change occurred. These *communication* features are well-understood, cheap and effective means for a simple work flow support [9].

4.2 Auditing

Every SCM system provides features to inquire the change history of specific configuration items; these features are mature and widely used. A yet unsolved problem is the traceability of *related documents:* How does one trace a change in implementation back to the design and back to the requirements? How is a change in the implementation related to a change in the documentation? Although *change-based versioning* or *activity-based SCM* [21] allows these changes to be associated with each other, there is still room for improvement here.

4.3 Accounting

Accounting facilities let users (and managers) inquire about the status of a product. SCM systems at least allow classifying components and versions according to specific properties (i.e. experimental, proposed, or stable); it may well be this simple tagging method is already sufficient. Again, we know of no research that has addressed pending problems in this area.

4.4 Controlling

Access control is one of the fundamental principles of automated SCM. Every SCM system features some kind of access control, typically via *locks* (only one user at a time can edit a file). Several SCM systems also support *access control lists* (only specific users are allowed to do changes); others rely on the security features of the underlying repository. Access control is widely used; it has never been a SCM research topic.

Tracking of change requests and defect reports is at the heart of the maintenance process, starting as soon as independent testing begins. The process of handling these, especially responsibility for decisions and definition of records to be kept, determines the responsiveness of an organization on user needs. In small organizations, a simple Excel sheet will provide enough support; bigger organizations require an elaborated data base with dedicated queries.

Advanced SCM systems like LIFESPAN [35] offer an elaborated management of *change requests;* in fact, the whole development process is organized along the processing of change requests. Although the effectiveness of the process remains to be validated, improvements are more likely to come from SCM vendors than from SCM researchers.

An important SCM topic is the tracking of *product defects,* as it provides immediate insight on the current product quality. Bug-tracking tools frequently come as standalone tools, from the freely available GNATS [23] to elaborated commercial systems. However, the integration with SCM repositories as well as automated testing facilities still leaves to be desired—a challenge for SCM vendors and researchers.

5 Conclusion

SCM is a mature discipline. It is mature in practice, as it is successfully used. And it is mature in research, since there is much to be taught—and not so much left to be researched. The only two research areas that are considered to have high potential are automated change integration and deployment issues; major improvements are also feasible in wide area connectivity, version management of structures, system modeling, consistency issues, lifecycle support, and integration issues.

Although several well-understood solutions are available, no single SCM system offers all solutions at once. Integration and flexibility are thus still issues for SCM users and SCM vendors—maybe also for SCM researchers, provided they find a way to validate the practical benefits of new SCM models.

Validation is also an issue for this paper, and the state of SCM in general. Upon compiling the tables, it was amazing to see how few hard facts were available to back specific judgements. The most important result of our assessment is that many more SCM experience reports and experiments are needed—*we need to know what we know before we can move on.* We thus encourage the SCM community to prove us right or wrong and look forward to fruitful discussions.

Acknowledgments. We thank the participants and organizers of the Dagstuhl Workshop on Software Engineering Research and Education [7] for their suggestions and contributions. Gregor Snelting provided useful comments on an earlier revision of this paper. Walter F. Tichy initiated the discussion on the state of SCM and helped a lot in ranking the individual SCM solutions.

References

1. BERLINER, B. CVS II: Parallelizing software development. In *Proc. of the 1990 Winter USENIX Conference* (Washington, D.C., 1990).
2. BINKLEY, D., HORWITZ, S., AND REPS, T. Program integration for languages with procedure calls. *ACM Transactions on Software Engineering and Methodology 4*, 1 (Jan. 1995), 3–35.
3. BUFFENBARGER, J. Syntactic software merging. In Estublier [8], pp. 153–172.
4. BURROWS, C., AND WESLEY, I. *Ovum Evaluates: Configuration Management*. Ovum, Inc., Burlington, MA, 1999.
5. CONRADI, R., AND WESTFECHTEL, B. Version models for software configuration management. *ACM Computing Surveys 30*, 2 (June 1998), 232–282.
6. DART, S. Concepts in configuration management. In Feiler [11], pp. 1–18.
7. DENERT, E., HOFFMAN, D. M., LUDEWIG, J., AND PARNAS, D. L. Software engineering research and education: Seeking a new agenda. Workshop Report 230, Dagstuhl, Feb. 1999.
8. ESTUBLIER, J., Ed. *Software Configuration Management: selected papers / ICSE SCM-4 and SCM-5 workshops* (Seattle, Washington, Oct. 1995), vol. 1005 of *Lecture Notes in Computer Science*, Springer-Verlag.
9. ESTUBLIER, J., AND CASALLAS, R. The Adele configuration manager. In Tichy [30], ch. 4, pp. 99–133.
10. ESTUBLIER, J., FAVRE, J.-M., AND MORAT, P. Towards scm/pdm integration? In Magnusson [20], pp. 95–106.
11. FEILER, P. H., Ed. *Proc. 3rd International Workshop on Software Configuration Management* (Trondheim, Norway, June 1991), ACM Press.
12. FELDMAN, S. I. Make—A program for maintaining computer programs. *Software—Practice and Experience 9* (Apr. 1979), 255–265.
13. FOWLER, G., KORN, D., AND RAO, H. *n*-DFS: The multiple dimensional file system. In Tichy [30], ch. 5, pp. 135–154.

14. FRÜHAUF, K. Hygiene in software works—Software configuration management. In *Proceedings of the Second European Conference on Software Quality* (Oslo, 1990), pp. 1–17.
15. HEIMBIGNER, D., AND WOLF, A. Post-deployment configuration management. In Sommerville [26], pp. 272–276.
16. HORWITZ, S., PRINS, J., AND REPS, T. Integrating noninterfering versions of programs. *ACM Transactions on Programming Languages and Systems 11*, 3 (July 1989), 345–387.
17. HUNT, J. J., VO, K.-P., AND TICHY, W. F. Delta algorithms: An empirical analysis. *ACM Transactions on Software Engineering and Methodology 7*, 2 (Apr. 1998), 192–214.
18. LEBLANG, D. B. The CM challenge: Configuration management that works. In Tichy [30], ch. 1, pp. 1–37.
19. MACKENZIE, D., AND ELLISTON, B. *Autoconf—Creating Automatic Configuration Scripts*. Free Software Foundation, Inc., Dec. 1998. Distributed with GNU autoconf.
20. MAGNUSSON, B., Ed. *Proc. 8th Symposium on System Configuration Management* (Brussels, Belgium, July 1998), vol. 1349 of *Lecture Notes in Computer Science*, Springer-Verlag.
21. MICALLEF, J., AND CLEMM, G. M. The Asgard system: Activity-based configuration management. In Sommerville [26], pp. 175–186.
22. OBJECT MANAGEMENT GROUP. *The Common Object Request Broker: Architecture and Specification*, Aug. 1991.
23. OSIER, J. M., AND KEHOE, B. *Keeping Track: Managing Messages With GNATS*. Cygnus Support, 1996.
24. PLOEDEREDER, E., AND FERGANY, A. The data model of the configuration management assistant. In *Proc. 2nd International Workshop on Software Configuration Management* (Princeton, New Jersey, Oct. 1989), W. F. Tichy, Ed., ACM Press, pp. 5–13.
25. ROCHKIND, M. J. The source code control system. *IEEE Transactions on Software Engineering SE-1*, 4 (Dec. 1975), 364–370.
26. SOMMERVILLE, I., Ed. *Proc. 6th International Workshop on Software Configuration Management* (Berlin, Germany, Mar. 1996), vol. 1167 of *Lecture Notes in Computer Science*, Springer-Verlag.
27. STALLMAN, R., AND MCGRATH, R. *GNU Make—A Program for Directing Recompilation*, 0.48 ed. Free Software Foundation, Inc., 1995. Distributed with GNU Make.
28. TICHY, W. F. RCS—A system for version control. *Software—Practice and Experience 15*, 7 (July 1985), 637–654.
29. TICHY, W. F. Smart recompilation. *ACM Transactions on Software Engineering and Methodology 8*, 3 (July 1986), 273–291.
30. TICHY, W. F., Ed. *Configuration Management*, vol. 2 of *Trends in Software*. John Wiley & Sons, Chichester, UK, 1994.
31. VAN DER HOEK, A., HEIMBIGNER, D., AND WOLF, A. L. Does configuration management research have a future? In Estublier [8], pp. 305–310.
32. VAN DER HOEK, A., HEIMBIGNER, D., AND WOLF, A. L. System modeling resurrected. In Magnusson [20], pp. 140–145.
33. WESTFECHTEL, B. Structure-oriented merging of revisions of software documents. In Feiler [11], pp. 86–79.
34. WESTFECHTEL, B., AND CONRADI, R. Software configuration management and engineering data management: Differences and similarities. In Magnusson [20], pp. 95–106.
35. WHITGIFT, D. *Methods and Tools for Software Configuration Management*. John Wiley & Sons, Chichester, UK, 1991.
36. ZELLER, A. Smooth operations with square operators—The version set model in ICE. In Sommerville [26], pp. 8–30.
37. ZELLER, A., AND SNELTING, G. Unified versioning through feature logic. *ACM Transactions on Software Engineering and Methodology 6*, 4 (Oct. 1997), 398–441.