Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 65 (2007) 215–221
Science of Computer Programming
www.elsevier.com/locate/scico

Software Configuration Management

Software Configuration Management (SCM) concerns the control of the evolution of complex software systems. CM is one of the most successful Software Engineering research fields with respect to acceptance in the industry. Almost any development organization that releases software applications and/or supplies complete software intensive systems, can only survive with the right attention to Configuration Management. The importance of SCM is also addressed by software process improvement models, such as CMM, CMMI and SPICE, in which SCM processes play a major role in achieving an initial level of maturity.

During the last three decades, SCM has emerged towards a mature Software Engineering discipline. During these years, several topics in SCM have been researched and their results are currently common in the available SCM tools. Software development is changing, which implies that aspects of SCM should correspondingly change over time to serve new software development needs. For example, model driven engineering is a growing field that also requires different SCM support. Another example of concern is the growth of software code bases that require extra tool support during maintenance (in order to find the correct information). In this special issue of the Science of Computer Programming we present two excellent papers that are related to given examples in new areas of research.

In this introduction, we provide a short overview of SCM. Furthermore, we present SCM in the perspective of the changing world of Software Engineering. Challenges in SCM are discussed to trigger new research in the SCM field. The SCM workshop is the main workshop in this field, so we provide a short historical overview of this also.

1. Overview of SCM

SCM is subject to evolution. The history of SCM development follows that of software development. In the 1960s, when software consisted of monolithic programs that were mostly implemented as one source modules, there was no apparent need for SCM. The software engineering community was focused on the art of programming—the production of effective algorithms occupying as little memory as possible. In the 1970s and the 1980s, software became more complex; programs were built in two stages, compilation producing binary modules from the source code, and linking combining binary modules in a program. The first generation of SCM tools appeared during this period: Source Code Control System (SCCS) and Make [5], covering the basic SCM disciplines: configuration and build management, and version management. Later, an improved version of SCCS, Revision Control System (RCS) [11] was developed. At this time, SCM was focused on "programming in the large" (versioning, rebuilding, and composition) [4]. RCS, Make and their different clones or variants, or tools using them, have dominated the SCM market for many years, and the principles they introduced are still widely used in the majority of modern tools. Make exists today in many forms (such as imake, gnumake, different "project-files", etc.) and RCS is used as a basis for many other, more advanced SCM tools. During the 1990s, the focus of SCM moved to the "programming in the many" [4], i.e., with emphasis on teamwork (process support, concurrent development and concurrent engineering in general). Change management, workspace management and process support became the new purposes of SCM tools. The complexity of SCM tools increased as software became more complex. Several new advanced SCM tools that were very complex and very expensive appeared at this time. In addition to their complexity, they were not easy to introduce into the development process. For this reason, many companies developed their own systems, or used simple SCM tools, executing other SCM procedures manually.

0167-6423/$ - see front matter
doi:10.1016/j.scico.2006.10.003

The use of SCM increased significantly during the nineties for several reasons. The need for tools capable of managing software with growing complexity became obvious. At the same time, software development became more important in business, and many new software companies appeared in the market. Finally, software development focused on development processes influenced by the well-known Capability Maturity Model (CMM) from the Software Engineering Institute (SEI) [1], which pointed out SCM as an important ('key process') area. CMM defines an SCM process as including a number of activities and their planning (with emphasize on the planning) during the entire product life cycle.

As the software industry continues to grow rapidly, producing larger and more complex software, the need for software management also grows. One example is the increasing tendency for persons involved in the development of a system to be geographically dispersed. The new focus of software development is therefore on "programming in the wide" (web remote engineering, distributed development). Another example is the integration of SCM functions in different Integrated Development Environment tools, or in particular applications, in which the SCM functions were integrated as underlying activities in a development process.

2. SCM functions

Over the years, the scope of Software Configuration Management has been extended due to the additional requirement to control the evolution of software. SCM can be divided into a number of functions. Version Management is the most well known function, and a lot of people associate SCM only with Version Management, which is not wholly true. The storage of different versions of software artifacts in a repository is of major importance; for example, when bugs have to be solved in an earlier release of the product. All SCM tools provide Version Management, for example CVS [3], Subversion [9], Telelogic CM/Synergy [10] and IBM Rational ClearCase [7]. Most SCM tools build Version Management on top of an Operating System's file system; extra information about versions, like the software's creator and artifact's modification reason, are stored in a separate database.

Software Configuration Management is more than Version Management. Different functions are distinguished [2], and we will shortly describe some of them. Configuration Management or Configuration Selection is a special part of SCM that, in combination with version management, makes it possible to select particular versions of artifacts that together build a consistent set, a configuration. SCM provides tools for configuration using different versions of selection criteria, as the simplest "latest version", or named versions, or version that include particular changes, etc.

Build Management concerns the creation of the actual product, like compiling source files, linking object files into executables, and assembling final documents from a set of information files. The most well known tool in the field of build management is Make, which currently exists in many flavours in various SCM systems.

Release Management, another SCM function, concerns all activities that identify a system in order to be able to refer them to source versions when bug fixes have to be made on the older releases. *Installation Management* (sometimes assumed as a part of Release Management) concerns all activities that install a system at the customer's site given a released (and packaged) system. *Workspace management* concerns all activities that provide engineers, integrators and any other development persons (roles) the proper "views" about the product at hand. Selection of the proper versions are more or less hidden for the developer. There are more functions in SCM, which we will not discuss here.

Change Management keeps track of all the changes in a product in the development and maintenance phase. The reason for a change can be the correction of an error, the improvement of a component, or the addition of functionality. Change management is often supported by separate tools that are integrated with the main SCM tool. Examples of such tools are Serena PVCS tracker [8], Synergy/Change [10], and ClearQuest [6]. Change management has two main objectives: the first is to provide support for the processing of changes, including change identification, analysis, prioritizing, planning for their implementation, decisions for their rejection or postponement, decisions for their integration in new product releases. The second objective is traceability, i.e., keeping track of all active and implemented changes.

The above functions are described in terms of their SCM functionality. In [4], the author identifies three main services as provided by SCM systems:

- Manage repositories (keeping track of versions and composing configurations)
- Help engineers in developing products (workspaces, cooperative development)
- Control and Support of the development process (process model and activities)

Managing repositories includes keeping track of all artifacts that are used in software development and maintenance, their versioning, and their organizations and configurations in the form of projects and products. While version management provides a high functionality on the meta level, the configuration and structuring of artifacts has always been a weak side of SCM. SCM has never managed to specify principles for structuring complex products, nor has it been able to define a product information model.

Engineering help in development products is mainly focused on (a) managing repositories automatically in the context of an integrated development environment and (b) providing support for a concurrent development, i.e., enabling parallel and distributed activities, increasing awareness of the on-going changes and synchronizations of the different versions of artifacts. Although there exist several SCM tools providing advanced support, distributed and concurrent engineering is still far away from full support.

The Control and Support of the development process includes support for a set of activities related to project stakeholders. An example of these activities is Change Management and quality assurance procedures.

3. SCM in software engineering perspective

In recent years, software engineering has changed and SCM has reacted correspondingly to these changes. SCM is an old engineering field, and many researchers have brought their knowledge to the various subfields of SCM. Since the early eighties, many tools have been developed to support SCM. Among them commercial tools are available that provide a lot of SCM's functionality.

We see however, that some gaps in the SCM systems are currently present. SCM methods and SCM tools should be enhanced to support other trends in software engineering and system engineering. Here, we will envision some directions in which system and software engineering are moving, and their corresponding impact on configuration management.

3.1. From one to multiple projects

A software organization often runs many software projects in parallel. Different needs of customers are developed in separate projects to avoid interference. To decrease the time to market, organizations choose to develop software in parallel projects. It is, however, usually the intention to merge the developed features in a future release. Research on software merging has been performed for several years, but merging features have more impact than just merging software code; e.g. the code has to be tested and manuals have to be merged in a friendly way. It is therefore required that special measures be taken to better control parallel projects (that eventually merge into a single product).

3.2. From one to multiple disciplines

Another observation that stems from the embedded systems world, is that software and systems development are more and more integrated. Earlier on, hardware did not change that fast, but that has been changed! What is the difference between a C code and a VHDL description? Changing a hardware description may require a software change as well. That means that baselines will go across the borders of a discipline. Therefore, hardware and software configuration management systems should be more integrated. Similar arguments also hold for other pieces of hardware. Take, for example, the test of a system. The right software (versions) and hardware (version) should be in place. When there are different configuration management systems, their integrity has to be dealt with in another way.

3.3. From one to multiple products

Product Line Engineering has become common for many organizations to develop different variants of their products effectively. SCM systems should provide dedicated support for Product Lines. In the research field of Product Lines, SCM is often overlooked.

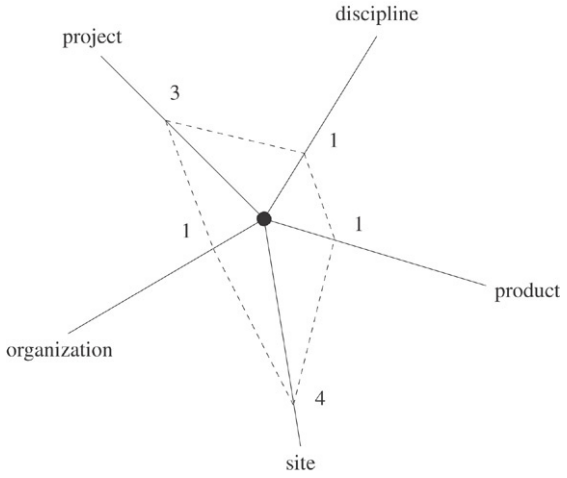

Fig. 1. Multiple directions in configuration management.

3.4. From one to multiple organizations

Due to acquisitions in companies and due to better distribution Development worldwide, one notices that more sites are involved in a single product (line) development. Most SCM systems support synchronization of data, which means that source code is visible for almost anyone anywhere, just after checking in a file.

3.5. From one to multiple sites

Within a single site, more organizations may be involved in the development of a product (line). This means that sources have to be shared between internal departments, which is often a topic for discussion. The need for supporting multiple organizations in SCM systems is becoming more and more important.

We summarize these thoughts in Fig. 1: the axes show the various perspectives, but are not necessarily orthogonal.

The different axes have value to different extents in companies. For example (marked as a dashed line in the figure), a company develops at four development sites, a single software application (one discipline, one product), having three projects in parallel development for different consecutive releases of the product in a single organization.

4. Recurring challenges for SCM

In addition to the SCM challenges related to systems engineering presented above, there are certain challenges which repeatedly occur through the entire history of SCM, and which look to be difficult to solve in the near future, since (so far) there are no general solutions to them.

4.1. Semantics of SCM assets

SCM deals with many different types of assets: software items, versions, configurations, changes, baselines, releases—to mention some of them. The main problem in SCM is that these assets are not formally defined, and their semantics are only understood on an intuitive level. Take, for example, version (as nicely discussed in [4]): what makes two versions of the same object? Obviously there is something which is common, and something which differs. SCM does not have a proper answer for what is common and what is different. In typical solutions, two versions of the same object have the same name, and the differences are expressed in differences in lines of code. This is far from expected! The core of the problem is that the semantics of SCM assets are not separated from semantics of the objects which an SCM manages. For example, in the case of object-oriented development, a proper level of versioning would be dealing with versioning of classes, operations, method implementations, and attributes. In a case of a document, versioning would deal with changes in format, or paragraphs, chapters, tables of contents, etc. In the versioning of a

design, UML assets would be the items for versioning. In databases, these will be relations, or records. We realize that versioning has different semantics in these cases, and it must be implemented in a completely different way for different objects. This is the immanent problem of SCM. One possibility is to implement support that is specific for types of objects. Indeed there exist such solutions—and usually they are integrated within other tools that process these objects. Examples of these are versioning and compare tools in MS Word. However, such solutions produce unsolvable problems for SCM: how to build a configuration of a system that includes different types of objects and differentiations between versions (where building a configuration essentially includes identification and extraction of particular versions of objects, and then putting them together)? To make it possible, SCM must integrate the tools which include different versioning mechanisms (which in most of the cases is not possible), or to build in semantics of the objects in the SCM (which is impossible to do in a general way). Another approach is that SCM builds its own mechanisms, which either are not appropriate (such as expressing differences in differences of lines), or which are too coarse grained (such as providing simple information—"the objects are the same, or the objects differ").

4.2. Information models

In some engineering tools, such as PDB, or CAD/CAM systems, information models (i.e. information structures) are related to product structures, as specified by different standards and described by dedicated languages. This makes it possible to provide the required support in managing assets. SCM does not have one information model—it has many, but they are not formalized and not standardized. The main reason for this is the different forms or assets, structures, and consequently information models of software products: there is a difference between development structure, design (logical) structure, implementation structure, packaging structure, and run-time structure. SCM only partially distinguishes these structures, and does not include rules for transformations and relations between them. This makes it more difficult to provide support for SCM functions (such as configurations, versioning, and collaborative development), which becomes apparent in the development paradigm "from one to many".

4.3. To integrate or to be integrated

Most of the SCM supporting functions are of an "administrative" nature—they keep control, or provide the proper information. They are not a part of a creative development/design process, and they are not of primary interest to the engineers; the tool should simply work, and remain invisible. For this reason, there is a strong requirement that these functions are well integrated with different tools (design tools, implementation, analysis and test tools, etc.). This integration remains a challenge; it is debatable whether the loose SCM functions (such as version management) should be a part of a tool, whether an SCM tool should communicate efficiently with these tools and automatically perform the activities triggered from the tools, or whether an SCM tool should provide an integrated development framework that integrates all development tools. The diversity of tools (and again the "from one to many" paradigm), makes this dilemma unresolvable today. In practice, and in research, we find all approaches; the integrated approach in the eighties (with PCTE—Portable Common Tool Environment) or more recently (Microsoft Team Foundation Server), the introduction of interoperability standards (such as WebDav), and the many implementations of Integrated Development Environment that use RCS or CVS tools.

The integrated approaches have the highest ambitions and claim a theoretical possibility to systematically provide SCM support. However, they have the problem of the integration of different tools (which in practice means that such integration, outside particular sets of tools combined on an ad hoc basis, is ignored).

5. SCM workshops and symposia

A series of SCM workshops and symposia, starting in 1988 and concluding with SCM 12 in 2005, mirror quite exactly the state of the art and of practice during this 17-year period. The SCM workshops have, from the very beginning, successfully attracted researchers, SCM tools providers, and to a large extent, SCM users. This fortunate combination has led to closing the gaps between researchers with new ideas, tool vendors with implementations, and customers providing the feedback from their experiences. In several cases, the vendors have are also researchers. All this has led to a fast expansion of SCM, and its very successful implementation and commercialization. The first workshops were more internal in character, and they established a stable SCM community. Later, from the fifth to the

eighth event (when workshops were transformed to symposia, thus giving more attention to dissemination of results, and gathering experience), SCM became known in a wider community. During that period, SCM become a standard part of a development process in most software and "software-intensive systems" development companies. This establishment has happened partially because of the success of the SCM workshops and symposia, but also partially due to the growing popularity of CMM (Capability Maturity Model), and a rapid increase of software complexity. This was the"golden age" of SCM. As the fundamental functions of SCM (version management, configuration and build management) have become available, either in SCM tools or integrated in other tools (Integrated Development Environment), the influence of the SCM workshops has ceased. At the same time, researchers have tried to find new challenges and new solutions for SCM. For example, SCM 10 was dedicated to new areas in SCM. Although it was shown that there is a need for SCM or SCM-like support in many new engineering domains (such as Web-engineering or Product Data Management), and there is a need for a successful integration with different technologies (for example object-oriented technologies, or component-based technologies), the further development of SCM alone has not shown as much strength as it did before. At the same time, it has become clear that SCM challenges have moved to other engineering domains as integrated parts of these domains. The last two workshops have shown more of the state of the practice and the state of the art—they have not introduced any novel approaches. SCM 12 was the last workshop in an amazingly successful series, and probably there will be no new SCM-dedicated events in the near future. The list below gives all the workshops and associated publications that these workshops produced.

1. International Workshop on Software Version and Configuration Control (SVCC), Grassau, Germany, January, 1988
2. Second International Workshop on Software Configuration Management (SCM-2), Princeton, New Jersey, October 24, 1989 (ACM Press, Software Engineering Notes, Vol. 17, No. 7, Nov. 1989)
3. Third International Workshop on Software Configuration Management (SCM-3), Trondheim, Norway, June 12–14, 1991 (ACM Press)
4. Fourth International Workshop on Software Configuration Management (SCM-4), Baltimore, Maryland, 1993 (Springer, Lecture Notes in Computer Science (LNCS) 1005)
5. Fifth International Workshop on Software Configuration Management (SCM-5), Seattle, Washington, May, 1995 (Springer, LNCS 1005)
6. Sixth International Workshop on Software Configuration Management (SCM-6), Berlin, Germany, March, 1996 (Springer, LNCS 1167)
7. Seventh International Workshop on Software Configuration Management (SCM-7), Boston, Massachusetts, May, 1997, (Springer, LNCS 1235)
8. Eighth International Symposium on System Configuration Management (SCM-8), Brussels, Belgium, July, 1998 (Springer, LNCS 1439)
9. Ninth International Symposium on System Configuration Management (SCM-9), Toulouse, France, September, 1999 (Springer, LNCS 1675)
10. Tenth International Workshop on Software Configuration Mangement (SCM-10/SCM 2001), Toronto, Canada, May, 2001 (Springer, LNCS 2649)
11. Eleventh International Workshop on Software Configuration Management (SCM-11/SCM 2003), Portland, Oregon, May, 2003 (Springer, LNCS 2649)
12. 12th International Workshop on Software Configuration Management, Co-Located with ESEC/FSE 2005, Lisbon, Portugal, September 5–6, 2005 (ACM Press, SBN:1-59593-310-7)

In our opinion, SCM is still an important research topic. The industry is still facing problems which have not been resolved by research. Also the move towards product line development (instead of a single product development) requires other features of SCM systems. In industry, there is a lot of experience in keeping SCM alive in software development. We see, however, that this is mostly based on industrial experience through the years. Software engineers as delivered by universities have hardly any knowledge about SCM as established in industry. The training of software engineering students to make them aware of SCM is thus very important, although it is challenging, since it is not simple: keep the simplicity in order to gain understanding and introduce complexity via SCM, as this is what SCM is used for. We have already discussed the existence of other challenges. This governs our belief that although there will be no dedicated SCM events, problems, challenges and research activities around SCM will remain.

6. Special issue

During the 12th International Workshop on Software Configuration Management in 2005, we started working on a special issue on this topic. Elsevier was willing to support this idea, for which we are grateful. We would also like to acknowledge the support of Vrije Universiteit, Amsterdam (the Netherlands) and Mälardalen University (Sweden).

Many people have been involved in the creation of this special edition. We are especially grateful to the community effort represented by this special issue: the many authors making an effort and submitting in total eight papers, and the reviewers who produced their reviews within a tight time window in early 2006. We want to thank the reviewers: Geoff Clemm, Lars Bendix, Harald Gall, Serge Demeyer, Reidar Conradi, Bernard Westfechtel, Claudio Riva, Jacky Estublier, Jim Whitehead, Annita Persson Dahlqvist, Chris Lüer, Rainer Koschke, Michele Lanza and Magne Syrstad.

In this journal, we present two interesting papers. We start with "Visual Assessment of Software Evolution", in which the authors apply interesting visualization techniques to support managing software evolution.

In the second paper, "Odysssey-SCM: An integrated Software Configuration Management Infrastructure for Model-driven development", the authors elaborate on the fact that a file is not always at the right level of granularity to store software artifacts.

We hope this special issue provides you with a lot of new insights which help you forward in your research, your industrial practice or anywhere else.

References

[1] Carnegie Mellon University, Software Engineering Institute, The Capability Maturity Model, Addison Wesley, 1995.
[2] I. Crnkovic, U. Asklund, A. Persson Dahlqvist, Implementing and Integrating Product Data Management and Software Configuration Management, Artech House Publishers, 2003.
[3] http://www.nongnu.org/cvs/.
[4] J. Estublier, Software configuration management: A roadmap, in: Proceedings of 22nd International Conference on Software Engineering, The Future of Software Engineering, ACM Press, 2000.
[5] S.I. Feldman, Make, a program for maintaining computer programs, Software—Practice and Experience 9 (4) (1979) 255–265.
[6] Rational ClearQuest. http://www-306.ibm.com/software/awdtools/clearquest.
[7] http://www-306.ibm.com/software/awdtools/clearcase/.
[8] http://www.synergex.com/solutions/pvcs/products/tracker/.
[9] http://subversion.tigris.org/.
[10] http://www.telelogic.com/corp/products/synergy/index.cfm.
[11] W.F. Tichy, RCS—a system for version control, Software—Practice and Experience 15 (7) (1985) 637–654.

Guest Editors
René Krikhaar*
Vrije Universiteit,
Amsterdam,
The Netherlands
E-mail address: rkrikhaa@cs.vu.nl.

Ivica Crnkovic
Mälardalen University,
Sweden

Available online 20 November 2006

* Corresponding editor. Tel.: +31 20 59 87824.