Software Configuration Management
A Road Map

Jacky Estublier
DassaultSystèmes / LSR, Grenoble University,
Actimart, Allée de Roumanie
38610 Gieres France
jacky@imag.fr

ABSTRACT

This paper, in the first chapter summarizes the state of the art in SCM, showing the evolution along the last 25 years. Chapter 2 shows the current issues and current research work under way in the area. In chapter 3, the challenges SCM has to take up, as well as SCM future research are discussed.

Keywords

Software configuration management, Version control, process support, concurrent engineering, federation, interoperability, architecture.

1 WHAT IS SCM?

Current definition would say that ***SCM is the control of the evolution of complex systems***. More pragmatically, it is the discipline that enable us to keep evolving software products under control, and thus contributes to satisfying quality and delay constraints.

SCM emerged as a discipline soon after the so called « software crisis » was identified, i.e. when it was understood that programming does not cover everything in Software Engineering (SE), and that other issues were hampering SE development, like architecture, building, evolution and so on.

SCM emerged, during the late 70s and early 80s, as an attempt to address some of these issues; this is why there is no clear boundary to SCM topic coverage. In the early 80s SCM focussed in *programming in the large* (versioning, rebuilding, composition), in the 90s in *programming in the many* (process support, concurrent engineering), late 90s in *programming in the wide* (web remote engineering). Currently, a typical SCM system tries to provide services in the following areas:

Managing a repository of components. There is a need for storing the different components of a software product and all their versions safely. This topic includes version management, product modeling and complex object management.

Help engineers in their usual activities. SE involves applying tools to objects (files). SCM products try to provide engineers with the right objects, in the right location. This is often referred as workspace control. Compilation and derived object control is a major issue.

Process control and support. Later (end 80s), it became clear that a, if not the, major issue is related to people. Traditionally, change control is an integral part of an SCM product; currently the tendency is to extend process support capability beyond these aspects.

1.1 Short History

In the 80s, the first systems were built in house and focussed closely on file control. Most of them were built as a set of Unix scripts over RCS (a simple version control tool) and Make (for derived object control).

From this period we can mention DSEE [29], the only serious commercial product, which introduced the system model concept, an Architecture Description Language (ADL) ancestor; NSE [34] which introduced workspace and cooperative work control; Adele which introduced a specialized product model with automatic configuration building [15], and Aides de Camp (now TRUE software) which introduced the change set (see later).

The first real SCM products appeared in the early 90s. These systems are much better. They often use a relational database but still rely on file control, they provide workspace support, but no or built-in process support. This generation included Clear Case [30] (DSEE successor) which introduced the virtual file system and Continuus [5][12] which introduced, with Adele, explicit process support [20]. Continuus and Clear Case are currently the market leaders.

In the second half of the 90s, process support was added and most products matured. This period saw the consecration of SCM, as a mature, reliable and essential technology for successful software development; the SCM market was over $1 billion sales in 1998. Many observers consider SCM as one of the very few Software Engineering successes.

1.2 Component repository

Most SCM products are based on a tiny core of concepts and mechanisms. Here is a short summary of these concepts.

1.2.1 Versioning

In the early 70s, the first version control system appeared [33]. The idea is simple: each time a file is changed a *revision* is created. A file thus evolves as a succession of revisions, usually referred to by successive numbers (foo.1, foo.2 .). Then from any revision, a new line of change can be created, leading to a revision tree. Each line is called a branch (the branch issued from foo.2 is called foo.2.1, foo.2.2…).

At the same time, 3 services were provided: History, delta, multi user management and a bit later, merging facilities. History simply records when and who created a revision along with a comment.

Deltas were provided because two successive revisions are often very similar (98% similar in average)[22]. The idea is to store only differences (the 2% that are different). Of course, it vastly reduces the amount of required memory.

Multi-user management consists in preventing concurrent changes from overlapping each other. A user who wants to change a file creates a copy and sets a lock on that file (check-out); only that user can create a new revision for that file (check-in).

Despite the fact that all this is 25 years old, it is still the base of the vast majority of today SCM systems.

1.2.2 Product model

From the beginning, the focus was on file control. It is no surprise to see that, even today, the data model proposed by most vendors resemble a file system, plus a few attributes, often predefined. This is archaic and contrasts with today's data modeling.

1.1.1 Composition

A configuration is often defined as a set of files, which together constitute a valid software product. The question is twofold: (1) what is the real nature of a configuration, and (2) how to build it, prove its properties and so on.

Surprisingly, in most systems, a configuration is not an object, but "something" special. It is a consequence of a weak data model in which complex objects and explicit relationships are not available.

The traditional way to build a configuration is by changing an existing one. No correctness criteria are available.

In the change-set approach, a change, even it involves many files, receives a logical name (like "FixBug243"). Later on, a configuration can be produced as a set of change-sets to add or remove from a base configuration (like "C2 = C1 + FixBug243 – Extention2"), C1 being the base configuration and C2 the new one [23]. In the Adele system, a configuration is built interpreting a semantic description which looks like a query: the system is in charge of finding the needed components based on their attributes and their dependencies [16].

None of these approaches is available in the vast majority of today's systems [9].

1.3 Engineers support

Practitioners rejected the early systems because they were helping the configuration manager, and bothering everybody else. A major move toward acceptance was to consider the software programmer as a major target customer: helping him/her in the usual SE activity became a basic service.

1.3.1 Building and rebuilding

The aim of rebuilding is to reduce compilation time after a change, i.e. to recompile, automatically, only what is needed. Make [24] is the ancestor of a large family of systems based on the knowledge of "dependencies" between files, and their last modification date. Make proved to be extremely successful and versatile, but difficult to use and inadequate in many respects. All attempts to do substantially better have so far failed. Most systems "only" generate the makefiles.

1.3.2 Workspace support

A workspace is simply a part of a file system were the files of interest (w.r.t. a given task like debug, develop etc) are located. The workspace acts as a sphere where the programmer can work, isolated from the outside world, for the task duration. The SCM system is responsible for providing the right files (often a configuration), in the right file system, to let users work (almost) independently, and to save the changes automatically when the job is done.

It is this service that really convinced practitioners that SCM was there to help them.

1.3.3 Cooperative work support

A workspace is a support for concurrent engineering, since many concurrent workspaces may contain and change the same objects (files). Thus there is a need for (1) resynchronizing objects and (2) controlling concurrent work.

Resynchronizing, so far, means merging source files. Mergers found in today tools simply compare (on a line by line basis) the two files to merge, and a file that is historically common to both (the common ancestor). If a line is present in a file but not in the common ancestor, it was added, and must be kept; if a line is present in the ancestor and not in the file, it was removed and must be removed. If changes occurred at different places, the merger is able to decide automatically what should be the merged file. This algorithm is simply a heuristic that provides in output a set of lines with absolutely no guaranties about correctness. Nevertheless mergers proved to work fine, to be very useful and became almost unavoidable. See a survey in [4].

Controlling concurrent work means defining who can perform a change, when, on which attribute of which object. It is one of the topics of process support that, currently, no tool really provides.

1.4 Process support

Process support means (1) the "formal" definition of what is to be performed on what (a process model), and (2) the mechanisms to help/force reality to conform to this model.

A State Transition Diagram (STD) describes, for a product type, the legal succession of states (and optionally which actions produce the transition), and thus describes the legal way to evolve for entities of that type. Since SCM aims to control software product evolution, it is no surprise many process models are based on STDs. It is a product-centered modeling. Indeed, experience shows that complex and fine-grained process models can be define that way. Unfortunately, experience also shows that STDs do not provide a global view of a process, and that large processes are difficult to define using (only) STDs.

The alternative way to model processes is the so-called activity centered modeling, in which the activity plays the central role, and models express the data and control flow between activities [32]. This kind of modeling is preferred if a global view is required, if a large process is to be structured, or if products are not the main concern. But this approach lacks precision for product control. Experience has demonstrated that both are needed, but integration is not easy and the few tools that intended to do so only propose 2 independent modeling [31]. High level process models mixing both are not currently available in commercial products, but have been experimented [20].

1.5 Current state of practice, as seen by practitioners.

At SCM9 (September 99 in Toulouse France), attendees was asked to write down an answer to the 2 following questions: In your experience, (1) what are the most useful features, (2) what are the worse aspects, the most critical missing feature. It is interesting to note that the answers were pretty much consistent, coming from persons from different countries, different companies and using different tools. The answers were the following:

Most useful / appreciated features. Clearly the number 1 was change control, activity control, and workspace support. Then comes, in differing order: Global view, traceability etc.

Worse aspect, most missing feature. Clearly the number 1 was: Better and more flexible process support, concurrent and distributed engineering support. Then comes: scalability, efficiency, incrementality, cross platform capability, PDM compatibility, interoperability etc.

It is interesting to see that both the most appreciated and the most criticized feature concern process support. Almost no comments concerned the basic aspects of SCM, like versioning and merging. Practitioners think tools are good and stable enough but still lack efficiency, scalability and interoperability with the other SE tools.

It is likely that, in the near future, the distinctive features between tools will be, functionally, their strength in process support, technically, their capability to grow with the company needs to inter-operate with other company tools and to support concurrent, distributed and remote engineering.

2 CURRENT RESEARCH WORK

We can consider that SCM has reached a level of maturity, sufficient at least for practitioners to feel the tool really helps. Nevertheless, progress is still needed, but the major research activity has substantially changed. The following is an assessment of current research activity.

The work performed in SCM so far avoids being dependent on any programming languages and any semantic aspect of the software product. The strength of SCM comes from this attitude; strong and general tools were built relying only on trivial things like files, suffixes and lines of code. Many weaknesses of SCM come from the same: too little knowledge of the software product. The challenges in the future will be to increase SCM power, thus to increase the knowledge on the product, without having to pay too much (in complexity, efficiency, generality) for the knowledge of specific syntax (e.g. programming languages) and semantics.

2.1 Component repository

1.1.2 Ver*s*ioning

It was for long a major research topic in SCM. It seems no longer to be the case. In the 80s the major issues have been solved [28], including change set approaches and unified versioning i.e. all entities, including configurations, are typed and versioned in the same way [16]. Later on, it was shown that classic versioning and the so-called change oriented versioning can be unified and are complementary [10], [41]. In the future, all SCM tools will propose both.

The current trend is to make a clear separation between the mechanisms (the tree of branches and revisions for example) and the meaning of versions; the goal being to hide the low level mechanisms (branches and revisions). One of the strong points of the change set approach is that the user has no insight into the way versions are stored, he/she only knows the logical changes. Similarly, in Adele, 3 kinds of versioning have been defined, historical, logical and cooperative; each with different characteristics,

methods and behavior; but all implemented using (invisible) branches and revisions [17].

Still, the deep understanding of versioning is missing. Why are 2 objects versions of each other? At least they share something (but what), and at least they differ in some way (but which one?). The current answer is: they share a number of lines! A better answer to both questions is a prerequisite for a good version management.

2.1.1 Data model, Product model

The weakest aspect in current SCM tools is the data model they use (an attributed file system). In the 80s and 90s a number of works proposed interesting data models including O.O. entity relationship and versioning [2][3][17][25][28]. Unfortunately no major commercial database fit the needs, and building a specific DBMS does not seems reasonable, given the efficiency, scalability and reliability requirements. Building an advanced data model on top of a relational one (like Metaphase did [1]), which is flexible, efficient and easy to use is not simple, if possible at all.

Among the issues still under consideration, is where versioning is to be located: below the data model [25], as part of the data model [17], or on top of the data model. Of course, using a commercial DBMS imposes the third "choice". This is not the good solution.

A weak data model is a major problem because it hampers the other aspects, most notably process support, to reach a satisfactory level. Without good knowledge of the system to be managed, and thus a good model, no advanced SCM is possible.

The interaction between the database community and SCM was not sufficient to produce convenient DBMSs; and it is unlikely there will be a serious DBMS tailored for SCM purposes in the near future.

2.1.2 Composition, system models

Despite the work done in the 80s that has shown the feasibility of advanced configuration construction using configuration description languages [16] and later on the work on advanced system model [26][36], current tools do not use these features. There is still an important area of research here, at least to produce systems where these ideas can be put into practice.

A system model describes, at a high abstraction level, a software product in terms of its components and their relationships. A system model looks very much like an Architecture Description Language (ADL), or a Configurable Distributed System (CDS) formalism. This analogy has been analyzed in [39] and has led to the following:

	ADL	SCM	CDS
Composition	***	*	**
Behavior	*		
Consistency	**	*	*
Building		***	*
Versioning		***	*
Selection		**	

In this table, we roughly assess the capabilities of three different streams of work, all related to modeling software systems. The basic concepts of a typical ADL are components, connectors and configurations. ADLs try to define some behavior and consistency constraints. ADLs have a static view of software, whereas CDSs have a dynamic view i.e. how the system evolves and reconfigures at execution. They concentrate on the description of how components can be dynamically instantiated and connected.

Each category of system proposes software description formalisms; the number of dots (0 to 3) indicates the extent to which the formalism proposed is convenient for the description of each aspect. SCM systems are essentially good for building and versioning, while ADL focuses on composition and consistency, and CDS on dynamic composition.

It is interesting to see that each kind of system focuses on different aspects, and none covers all the needs. The use of 3 different languages for describing the "same" thing with substantial overlapping is a waste of energy and is highly error prone. Fundamental work is needed to define if a single language is enough or/and to define complementary and coordinated formalisms controlling in a clear way different aspects of a system (more likely). This is an essential research task that spans other major research areas in SE.

2.2 Engineer help

2.2.1 Building

The goal of a rebuilding system is first to minimize the number of recompilations after a change, second to propose a formalism describing the (compilation) dependencies, which is easy to use and maintain. In both respects Make is far from ideal. Many work has been done to propose a better system than Make. Currently over 30 different flavors of Make have been proposed and used, including Vmake, Imake, gnuMake and Odin [8][40]. Few original propositions have been made.

Language dependent systems have been proposed (smart recompilation): knowing the change and the real dependencies, allows us to reduce the number of files to recompile. Unfortunately the time required by source code analysis is often larger than the time saved by avoiding a few files from being recompiled. Here again, the limit of what can be done without syntactic or semantic knowledge seems to have been reached, and source code analysis is too expensive with respect to the expected advantage (after all, the issue is to reduce rebuild time).

The current solution is to generate the Makefiles based on the information the SCM tool has. It solves the issue related to formalism, not efficiency. There is a clear need to do better than Make; but it is a serious challenge.

2.2.2 Workspace support

The current view of a workspace is rather simplistic: a set of files and directories. What practitioners require is much more: they want a given (version of) a complex object, at least a configuration, to be available for their tools, at a given location.

Each SE tool requires "its" objects to be under a specific format. Most SE tools only know about files (editor, compilers), but a few require those files to be under specific names, directories and formats (many development environments) while others require objects to be in a DBMS under a given schema.

What is required are objects (i.e. containing literals, object references and other objects) that can be represented under the required format and at the required location (computer).

In current implementations, all workspaces are managed by a central DB, which limits scalability, impedes heterogeneity, and severely reduces efficiency and availability. We need truly distributed, heterogeneous and efficient workspaces, which means each workspace must be autonomous and rely on "its" own local store, not on a centralized SCM system [22].

2.2.3 Cooperative and remote work support

In SE, as well as in any engineering domain, the reduction of the product life cycle (a few months between releases), together with the increase in product complexity and team size have dramatically increased the pressure for more concurrent work; but concurrent work is poorly supported at present. The traditional locking schema has been replaced by file merging but not much more is available.

There is a need to be able to define, at high level, the cooperation strategies, the work organization and coordination, the procedures by which somebody, at a given point is time, is able to "see" an object, to change some of its attributes, not some others; to resynchronize with somebody, not somebody else and so on.

Fundamentally, concurrent engineering relies on the capability to *merge* pieces of work done in a concurrent way on the same object. Merging objects is thus a central issue, but objects contain attributes; merging objects means merging their attributes. Some attributes have an exact merge (like composition), others approximate (like source code), and others not at all. Currently, only file merging for source code, using the trivial line based heuristic technique is available. Unfortunately, syntactic or semantic mergers, while available for a long time (early 80s), are too slow to be used, at least for programs [4]. May be web-based applications can develop and use XML / Html mergers. Anyway, current mergers only deal with source code, which is only one of the many object's attributes; they are not object mergers. O*bject* mergers, as available for a long time in Adele, will soon become a reality.

These models and mechanisms must be able to model and control development strategies like "open source", virtual enterprise, and remote teams as well as in house critical software. Technically they must cope with distributed and remote site, efficiency, availability and confidentiality. This is another challenge [22].

2.3 Process support

SCM is the place where software process technology proved to be applicable and successful. For example, change control is a process that can be found in all tools; cooperative engineering is another specific process.

Nevertheless, it is a domain where there is considerable room for improvement. STDs as well as triggers are clearly only a tiny part of the answer. Activity based approaches, alone, are not sufficient. Good integration between the different approaches is still needed. It is a challenge to have both high level model (for concurrent activity, versioning issues and so on) and efficient process engine for their execution.

There is a need to define high level and specific formalisms in which the different aspects of SCM processes can be modeled, executed, tailored, measured, and enhanced. Today solutions are partial, low level, and often inadequate. Major research work is needed in this area.

3 SCM CHALLENGES

The presentation above, as well as most work in SCM research, has a major flaw. It considers SCM in isolation; SCM tools are too often monolithic and their capability to inter-operate is limited. During the last years, things have dramatically changed; SCM tools are one among a growing number of other tools and thus are challenged in a number of dimensions.

3.1 Functional challenges

The first challenge is to address the issues raised above, namely to find or to put into production advanced concepts and mechanisms for

- data model, with complex objects, explicit relationships, files,
- versioning, homogeneous for all type of objects and relationship,
- configuration control, with automatic selection and consistency criteria,
- rebuilding, multi-platform, efficient with high level formalism,
- work spaces, distributed, scalable, heterogeneous and autonomous,
- concurrent engineering with high level models and scalable solutions,
- process control with multiple views, hybrid approaches and high level models.

3.2 Efficiency, scalability, availability.

All the above issues are by themselves challenging, but it is of critical importance to understand that a (the?) major limitation today is related to efficiency. Checking out a configuration, or labeling it, takes from minutes to hours, rebuilding takes from hours to days; dynamic selection or controlling concurrent engineering slows down substantially the work and so on. Computing time is an important overhead in software development, and practitioners do not accept to be slowed down any further.

The real challenge is to find novel, powerful and elegant solutions that are highly efficient, available, reliable and scalable. In any of the above bullets, these characteristics must be given top priority.

3.3 Data management

A large number of tools and environments manage a local data repository, sometimes versioned. Usual examples include a programming environment or 4GL tools. The difficulty is to find a way to make all these repositories inter-operable in order to avoid (too much) duplication and inconsistency. The real challenge is to make compatible (enough) the many different definitions, formats and versioning approaches. We have known for a long time this is a tough issue.

3.4 Process management

A growing number of tools include a process, in an explicit formalism or not; they are called Process Sensitive Systems (PSS). Today a large number of PSSs are present in industry, like planners, mail systems, GroupWare, workflow, project managers, business tools and so on. The data and concepts on which PSSs and an SCM system operate are roughly the same (task, time, resource, and data). From change control up to marketing strategies, the company processes are forming a continuum with large overlap between the processes managed by the different PSSs. Companies will require to cover the complete process spectrum, with minimum redundancy and overhead, but sill using the specialized tools they are used to. SCM is nothing else than one of these.

There is fundamental work to do to make PSSs inter-operate in a clear and clean way.

3.5 PDM vs. SCM

The definition of SCM says “the control of the evolution of complex systems”. This covers many engineering disciplines, not only software engineering. Indeed other type of engineering -electrical, mechanical, 3D design and so on- have developed their own tools, with leaders like Metaphase or Sherpa [1][14][28]. Let us call these tools Product Data Management (PDM) tools.

Both domains look very similar, but have at least two fundamental differences. First, in PDM, there is a clear distinction between the design (e.g. a bicycle design) and the product itself (a given bicycle). In software the design (the program) and the product (the software product) are almost the same; the later can be derived from the former at almost no cost. Second, in PDM, the product (will) have a physical existence, which confers unquestionable (spatial, mechanical) properties and constraints. For that reason, in PDM, the main structure is always its part structure (a bicycle has 2 wheels, a frame). In software there is no such “obvious” real structure; parts are arbitrary abstractions with loose relationships. Perhaps for these reasons product models in DPM are more advanced than in SCM[21].

Unification of both fields is needed because software constitutes a growing part of almost any industrial product. Unfortunately, vendors are not headed in this direction, because of deep underlying difficulties, and for efficiency and usability reasons.

3.6 Web support

The web had two effects on SCM: local, distributed and remote development can be made similar (see the workspace and scalability discussion). Distributed work space control will involve creating an infrastructure for the management of multiple copies of objects, with the distinctive feature that they can have different values, different formats, and can (must) be resynchronized. This kind of service is required by any application dealing with concurrent engineering and thus deserves to become a standard service on the net; for instance a CORBA service. Http extensions (WebDav) [7][45] are going in that direction.

On the other hand, the web produced a new kind of artifact: the web pages. Web pages are released but evolving products, containing essential pieces of information, and highly related with other pages. They clearly require configuration control. But when comparing a web page with a source file, the differences are easy to notice: The number of pages is 100 times larger than the number of files, the time between changes is 10 times shorter, the number of contributors is 100 times larger, they are not computer specialists and so on [13].

Solutions used for software cannot be used as they are. New ideas and techniques have to be invented, new tools to be built. This market is booming and solutions are urgently required. Many observers estimate this market is even now larger than software development, and it increases much faster ($5 billion sales are expected by 2002 (Marry Lynch co)). Products coming from other communities like document managers are entering this promising market. Even if SCM vendors have some experience it is unclear who will take that huge market.

3.7 Growing number of features

SCM tools, as many others, are facing the problem of becoming too big. The tendency is to include more and more services, and to (try to) address more and more domains. In recent years distributed development has been integrated, then remote development, then change control, then process support, then Web support and so on. On other dimensions, architecture languages, dynamic reconfiguring systems, deployment and so on are also candidates for integration inside an SCM.

This is an indication of success, but also worrying. The challenge is to find a new architecture for SCM allowing it to break the systems into independent pieces, and to compose your SCM system from the features that are currently required. Having a low introduction cost, and the ability to evolve, extend and scale up with the company is a major challenge that no tool currently meets. This challenge can be addressed only by changing deeply the architecture and philosophy of SCM systems.

4 CONCLUSION

Despite the many limitations and expected improvements discussed in this paper SCM proved to be one of the very few successful software engineering technologies. Indeed, the market is booming, with over $1 billion sales in 1998 and has excellent perspectives since only about 25% of mainframes, 15-20% of workstations and 5-10% of development PCs currrently has today an SCM system. Forecasts are about $2.1 Billions sales by 2000 and $3.3 Billions by 2002 [35].

It may appear that most of SCM research was performed in the 80s, and only tool improvement can be expected in the future. I do not support this view; I have tried to identify the areas for further research in the usual SCM core topics. Even if few new ideas have been proposed and few serious experiments have been performed in recent years, I think that much is still to be done both to find new concepts and better implementations. Unfortunately, too often, this work requires heavy developments and experimentation, which currently dissuade most academic researchers.

The future of SCM research and tools is unclear. The basic services will become understood, mature and stable enough to be standardized. They will fall into the public domain, as basic services anyone can expect from a platform, for instance versioning, rebuilding, basic work space support and so on. Vendors will lose their control on these low level services.

Vendor added value will come from their ability to build, above this level, an SCM system providing core topic advanced services, targeted toward a specific client: a specific data model and versioning capability, specific concurrent engineering facilities and so on. A major change is that that second layer will be considered as a basic SE platform kernel, rather than a stand alone product. The issue will be to standardize and “componentize” SCM systems in a way allowing us to build easily, tailored and highly efficient solutions.

Indeed, above this kernel, will be plugged a number of specific tools dedicated to a facet of SE, like process support, concurrent engineering control, project support; or dedicated toward a specific application domain like Web support, PDM control, deployment, electrical or mechanical tools and so on. SCM research and vendors are currently starting to address all these issues, but with limited scope, and not necessarily with all the requisite expertise. On this layer, for each tool, it is unclear who will take the lead, not necessarily SCM vendors. The way all these tools will cooperate, to build a fully fledged, evolutive and efficient SE environment, is an active research topic (mega programming, COTS federations etc). This last issue, interoperability control, can become another area where SCM research and tools can contribute.

Nevertheless, in the near future, provided the number of core topic issues yet to be solved and the efficiency, scalability and usability issues, no one of these evolutions will be seriously addressed by SCM vendors. SCM tools will still grow, propose proprietary solutions and still consider SCM as an isolated domain. SCM research should take the opportunity that many SCM challenges are currently under work in other SE domains, to foster a synergy between these research domains and SCM, bringing in the experience and know-how that made the strength of SCM, showing a path toward the useful and successful tools software engineering needs.

REFERENCES

[1] 1 "A Comprehensive Configuration Management Solution, Metaphase Product Structure Manager and Advanced Product Configurator", Metaphase Technology, MW00206-A.

[2] A. Bjornersledl and C. Hullen. *Version control in an Object-Oriented Architecture*. In Won Kim and Frederick H. Lochowsky. editors. Objects-Oriented concepts, databases and application. Chapter 18, pages 451-485, Adisson-Wesley. 1990.

[3] E. Bratsberg. *Unified Class Evolution by Object Oriented views*. Proc of the 11th Conf on the relationship approach. LNCS N0645, Springer Verlag, Oct 1992.

[4] J. Buffenbarger. Syntactic Software Mergin. SCM5, Seattle June 1995, pp153-172. Springer Verlag LNCS 1005.

[5] M. Cagan and A. Wright. *Untangling configuration management: Mechanism and methodology in CM systems.* In Proc. 4th Int. Workshop on Conf. Man. SCM4. Baltimore 1993. Springer Verlag LNCS, pp35-53.

[6] "*ClearGuide: Product Overview*". Technical report, Atria Software, Inc.

[7] G. Clemm. *Versioning Extensions to WebDav*. Rational Software. May 1999. http://www.ietf.org/internet-drafts/draft-ietf-webdav-versioning-02.txt

[8] G. Clemm. *The Odin System*. SCM5, Seattle June 1995, pp241-263. Springer Verlag LNCS 1005.

[9] R. Conradi and B. Westfechtel. "*Configuring Versioned Software Product*". In SCM-6 Workshop. pp. 88-109. Springer LNCS 1167. Berlin, March 1996.

[10] R. Conradi and B. Westfechtel. "*Toward an Uniform Model for Software Configuration Management"*. In SCM-7 Workshop.pages 1-17. Springer LNCS 1235. May 1997.

[11] S. Dami, J. Estublier and M. Amiour. *"APEL: a Graphical Yet Executable Formalism for Process Modeling"*. Automated Software Engineering journal, January 1998.

[12] S. Dart. "*Concepts in Configuration Management Systems*". Proc. of the 3rd. Intl. Workshop on Software Configuration Management. Trondheim, Norway, june, 1991.

[13] S. Dart. Content Change Management: Problems for Web Systems. In Proc SCM9, Toulouse, France, September 1999. pp1-17. Springer Verlag LNCS 1675.

[14] "*EDL/Metaphase, Overview*", Metaphase Technology, MW00200-A, 29 pages.

[15] J. Estublier. « *A configuration manager: The Adele Database of Programs* ». In Proceedings of the workshop on Software Environments for Programming-in-the-Large. Pages 140-147. Harwichport, Massachussets, June 1985.

[16] J. Estublier, N. Belkhatir. Experience with a data base of programs. Proc. ACM Sigsoft/Sigpal conf. On practical Software Development Environments, pp84-91. Palo Alto, Dec 9-11, 1986.

[17] J. Estublier and R. Casallas. "*The Adele Software Configuration Manager*". Configuration Management, Edited by W. Tichy; J. Wiley and Sons. 1994. Trends in software.

[18] J. Estublier and R. Casallas. *"Three Dimensional Versioning"*. In SCM-4 and SCM-5 Workshops. J. Estublier editor, September, 1995. Springer LNCS 1005, pp118-136.

[19] J. Estublier. *"Workspace Management in Software Engineering Environments"*. in SCM-6 Workshop. Springer LNCS 1167. Berlin, Germany, March 1996.

[20] J. Estublier and S. Dami and M. Amiour. *High Level Process Modeling for SCM Systems*. SCM 7, LNCS 1235. pages 81--98, May, Boston, USA, 1997

[21] J. Estublier and J.M. Favre and P. Morat. *Toward PDM / SCM: integration?*. In proc SCM8, Bruxelles, Belgium, July 1998. Springer Verlag, LNCS 1439, pp75-95.

[22] J. Estublier. *Distributed Objects for concurrent engineering*. In Proc SCM9, Toulouse, France, September 1999. pp172-186. Springer Verlag LNCS 1675, pp192-196.

[23] P. Feiler. *Configuration management models in commercial environments.* Technical report CMU/SEI-91-TR-7. SEI 1991.

[24] S. Feldman. *A program for maintaining computer programs*. Software practice and experience, April 1979.

[25] B. Gulla, E.A. Carlson, D. Yeh. *Change-Oriented version description in EPOS*. Software Engineering Journal, 6(6):378-386, Nov 1991.

[26] . Gulla, J. Gorman. *Experience with the use of a Configuration Language*. In SCM-6 Workshop, Berlin, Germany, March, 1996. Springer Verlag LNCS1167, pp198-219.

[27] J. Hunt, K. Vo, W. Tichy. *Delta Algorithms: An empirical evaluation*. ACM Transactions on Software Engineering and Methodology. È(2): 192-214, April 1998.

[28] R.H. Katz. *Toward a unified framework for version modeling in engineering databases.* ACM Computing surveys. 22(4): 375-408, 1990.

[29] D. B. Leblang. and G.D. McLean. *Configuration Management for large-scale software development efforts*. In Proceedings of the workshop on Software Environments for Programming-in-the-Large. Pages 122-127. Harwichport, Massachussets, Jume 1985.

[30] D. B. Leblang. "*The CM Challenge: Configuration Management that Works*". Configuration Management, Edited by W. Tichy; J. Wiley and Sons. 1994. Trends in software.

[31] D.B. Leblang. *Managing the Software Development Process with ClearGuide*. SCM 7, LNCS 1235. pages=66, 80, May, Boston, USA, 1997

[32] J. Micallef and G. M. Clemm. *"The Asgard System: Activity-Based Configuration Management"*. In SCM-6 Workshop, Berlin, Germany, March, 1996. Springer Verlag LNCS1167, pp175-187.

[33] M. Rochkind. *The Source Code Control System.* IEEE Trans. on Soft. Eng.. Vol SE-1, pp364-370, Dec 1975

[34] Sun Microsystem. Introduction to the NSE. Network Software Environment: Reference Manual. Sun. Part N° 800-2095, March 1988.

[35] OVUM. Configuration Management. 1999.

[36] E. Triggeseth, B. Gulla, R. Conradi. *Modelling systems with variability Using the PROTEUS Configuration Language*. In SCM 7, LNCS 1005. pp. 216-240, Seattle. May 1995.

[37] Walter F Tichy. *Design implementation and evaluation of a revision control system*. In Proc.6th Int. Conf. Software Eng., Tokyo, September 1982.

[38] Walter F. Tichy. *Tools for software configuration management*. In Proc. of the Int. Workshop on Software Version and Configuration Control, pp. 1–20, Grassau, January 1988

[39] A.van der Hoek and D.M. Heimbigner and A.L. Wolf. *Versioned Software Architecture.* ISAW3, nov, 1998, pages 73—76.

[40] R.C. Water. *Automated software management based on structural models*. Software Practice and Experience. 1989.

[41] D.W. Weber. *Change Sets versus Change Packages: Comparing Implementation of Change-Based SCM*. SCM7, Boston, May 1997, Springer Verlag LNCS 1235, pp25-36.

[42] L. Wingerd, S. Seiwald. *Constructing a large product with Jam*. SCM7, Boston, May 1997, Springer Verlag LNCS 1235, pp36-49.

[43] C. Went, B. Wonsierwicz. *Source Control + Tools = Stable system*. Proc. Fourth Computer Software and Application System Development. March 1983.

[44] Jim Whitehead. *Goals for a Configuration Management Network protocol*.In SCM9, LNCS 1675, pages 186-204, Toulouse September 1999.

[45] WebDav. *HTTP extentions for distributed Authoring.* RFC 2518. http://andrew2.andrew.cmu.edu/rfc/rfc2518.htm. February 1999.