Technical Report
CMU/SEI-92-TR-008

The Past, Present, and Future of Configuration Management

Susan Dart

Technical Report

CMU/SEI-92-TR-8
July 1992

The Past, Present, and Future of Configuration Management

Susan Dart

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the SEI Joint Program Office HQ ESC/AXS

5 Eglin Street

Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF, SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service / U.S. Department of Commerce / Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides acess to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential con tractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: 1-800-225-3842 or 703-767-8222.

The Past, Present, and Future of Configuration Management

Abstract: Automated support for configuration management (CM) is one aspect of software engineering environments that has progressed over the last 20 years. The progress is seen by the burgeoning interest in CM, many technical papers and conferences involving CM, a large number of CM tool vendors, and new software development environments that incorporate CM capabilities. This paper is about future issues affecting solutions to CM problems. To put the future into perspective, it is necessary to discuss the past and present situation for CM. The past evolves around CM systems built in-house and supplemented with manual procedures and policies for executing the CM functions. The present consists of a better understanding of CM, the beginnings of a common vocabulary for CM, existence of many third-party CM tools and environments supporting CM, and recognition that a single CM system does not solve all CM problems and that there is a need for better understanding of CM process support. The future involves technical, process-oriented, political standardization and managerial challenges. These include the need to provide for new CM requirements, understand the effects of advances in environments, deal with governmental requirements on contractors for using certain CM capabilities, and acquire more management commitment for resources in solving the CM problems of an organization. One way to start addressing these challenges is through the definition of a CM services model that provides a conceptual framework for all CM capabilities. As CM is examined more closely in relation to software engineering, it becomes evident that advances in software technology are needed to aid advances in CM technology.

1. Introduction

Automated support for solving configuration management (CM) problems exists. Typical kinds of CM problems have been described in [4, 5] and typical, automated parts of solutions can be found in [12, 7, 15, 39, 31, 32, 33]. A CM solution is dependent on an organization's needs and how it defines CM. The standard definition for CM taken from IEEE standard 729-1983 [30] includes:

- **Identification**: identifying the structure of the product, its components and their type, and making them unique and accessible in some form. For example, this addresses the question, "What version of the file is this?"

- **Control**: controlling the release of a product and changes to it throughout the life cycle by having controls in place that ensure consistent software via the creation of a baseline product. For example, this addresses the question, "What changes went into the latest version of this product?"

- **Status Accounting**: recording and reporting the status of components and change requests, and gathering vital statistics about components in the

product. For example, this addresses the question, "How many files were affected by fixing this one bug?"

- **Audit and review**: validating the completeness of a product and maintaining consistency among the components by ensuring that the product is a well-defined collection of components. For example, this addresses the question, "Are all the correct versions of files used in this current release?"

When examining current technology that automates CM functions, it becomes clear that the definition of CM as given by the IEEE standard needs to be broadened to encompass the extra functionality found in CM systems. This concerns:

- **Manufacturing**: managing the construction and building of the product in an optimal manner. For example, this addresses the question, "What versions of files and tools were used to generate this latest release?"

- **Process management**: ensuring the correct execution of the organization's procedures, policies, and life-cycle model. For example, this addresses the question, "Were all the files tested and checked for quality before being released to the customer?"

- **Team work**: controlling the work and interactions between multiple developers on a product. For example, this addresses the question, "Were all the locally made changes of the programmers merged into the last release of the product?"

The CM solution is pervasive—it affects the *software development environment*, the *software process model*, the *users* of the CM system, the *quality* of the software product, and the user's *organization*. Briefly, it affects the environment in terms of tool integration and functionality. That is, how tightly CM is integrated with the rest of the capabilities in the environment (e.g., Every time I do a "save" in the editor, is a new version stored?) and what CM functionality is provided (e.g., Do I need derived object pools and transactions?). The CM solution affects the process model and users of the CM capabilities in the sense of enforcing policies and procedures on the way users do their work and it keeps track (via an audit trail) of how the work is done. It affects the quality of the product in terms of the timeliness of its development and maintenance, e.g., the CM mechanisms will ensure that an inventory list is kept of all items that are in a release; thus, integrity is maintained which adds to the quality of the product. And because of this inventory, time is not wasted trying to determine what went into a release in case a bug needs to be tracked down. The CM solution also affects the user's organization in the sense that companies generally want a CM solution to be used globally, throughout the entire organization. In effect, the CM system acts as the central repository for most of the corporate software information and binds the organization to certain procedures and tasks. Thus, a CM solution has many ramifications that affect everyone in the organization as well as the way the organization does its business.

In general, a CM solution entails planning, defining a process, dealing with people, automat-

ing support, and making management decisions. The planning involves creating a CM plan that captures all the important aspects about doing CM. Defining a process means capturing all the steps, tasks, and associated policies and procedures needed for doing CM. Dealing with people means catering for different user roles that exist in the organization, such as the project manager, configuration manager, programmer, tester, quality assurance manager, and customer. Automating support is necessary to alleviate as much as possible the chores involved in doing CM, and it helps with maintaining the integrity and quality of the process and product. Making management decisions means deciding upon such issues as when to start using CM, whether to buy or build a CM system, and how to best perform technology transition for the CM system.

An organization must also address its solution from several perspectives. These perspectives are: corporate, project, developer, and application. The corporate perspective of CM is the organization's view and process of CM. Such a view is generally driven by the change management process. The project view is specific to a project group; for instance, each group may use a different CM system. The developer view is the individual programmer's and entails the specific functionality provided by a CM system. The application view concerns how CM is applied to a specific problem.

This article discusses the past and present situation concerning CM systems in order to focus on the future CM challenges. The past is characterized as in-house CM solutions whereas the present is characterized by many third-party CM solutions. The future involves technological, process-oriented, political standardization and managerial challenges. One way of addressing these challenges is through the definition of a CM services model, which is briefly discussed. This article concludes by raising some questions about the nature of CM in relation to software engineering problems in general.

2. The Past

In the past, CM was a private, organizational problem. A few organizations fell into a CM solution because they realized that managing changes to their software was very complex and they needed assistance. Other organizations did not use CM because the problems were not clearly evident nor were there any third-party tools available to assist them, and the cost of in-house development was viewed as too expensive. CM solutions generally entailed version control facilities that came with the operating system (such as SCCS [29] with UNIX) and some kind of build facility (such as job control scripts or Make [21]). Organizations were thus dependent upon the operating system vendor for any CM capabilities or had to develop them in-house. Usually change control facilities (such as change request forms and audit trails) were added. Along with a few automated facilities, most of the CM solution involved manual procedures and policies: people would keep any CM information in their head or filing cabinet; or a librarian would be assigned to carry out the CM functions; and in large companies, lengthy procedures and policies were documented in large company manuals.

While industry was finding its esoteric solutions, researchers were focusing their efforts to find third-party solutions, such as with RCS [28], Gandalf [17], and Cedar [9]. Generally speaking, the past brought about a good understanding of version control, compiling code, tracking and dealing with bugs, and a realization that simple version control was not the solution to CM needs.

3. The Present

The present is characterized mostly by technological issues. These include a better understanding of CM technology, such as the work done by the Software Engineering Institute (SEI) at Carnegie Mellon University and a recognition, acquired from practice and experience within the software engineering community, of the complexity of a CM solution.

3.1. Understanding CM Technology

The SEI has evaluated and assessed CM capabilities of environments, such as Rational [27], ISTAR [19], and NSE [22]; tools, such as IDE's Software Through Pictures and Procase's SmartSystem; and CM systems, such as ADC [1] and CCC [8]. We have also experimented with various CM tools, which resulted in the definition of a spectrum of concepts that captures the functionality found in CM systems [12] and a categorization of CM systems for developers [15]. In addition, we have examined the issues of integrating tools with CM capabilities [18] and are involved in standardization efforts, such as for environment frameworks [25], to ensure that CM is suitably addressed. Also, the SEI has identified process maturity levels for organization and a few key practices essential to carrying out CM [11]. We have analyzed the technology transition issues [16] that make it difficult to introduce new tooling into an organization. All in all, such work has led us to a better understanding of CM technology and how it fits with software engineering environments.

A third-party CM tool industry is flourishing which has taken many of the ideas from researchers and incorporated them into environments and computer-aided software engineering (CASE) tools. Much of the practical CM technology is coming out of this industry.

The term "CM technology" is meant to denote the automated capabilities providing CM functions and the infrastructure to support those capabilities, such as an operating system or environment. At present, CM technology exists as third-party tools [12, 7] or as capabilities within an environment [27, 19, 22]. The tools are sold either as a base tool that enables the user to manipulate its primitives to form a customized CM system, or they come as a turnkey CM system with preset controls, user roles, and policies.

When surveying the tools and environments, it is possible to extract a set of 15 CM concepts [12] that capture the essence of automated support for CM. These concepts are:

1. **Repository**: captures CM information and stores versions of files as immutable objects, as in RCS [28].

2. **Distributed component**: allows distributed users to have access to a repository where the CM facilities appear to span the network of heterogeneous workstations, as in Sherpa DMS [13].

3. **Context management**: captures in a domain-specific manner the working context for the user, thereby eliminating the need for users to remember how

they got to a particular working status and what all the data items, their relationships, and tools are in that context, as in Powerframe [24].

4. **Contract**: represents a formal plan for, and a record of, a unit of work on a configuration item, as in ISTAR [19].

5. **Change request**: assists in driving the process of change to configurations and keeping an audit trail of the changes, as in LIFESPAN [39].

6. **Life-cycle model**: represents the process of developing and maintaining configurations through various phases, as in CCC [8].

7. **Change set**: represents a logical change to a product and a means of creating any version of a configuration that is not necessarily dependent on the latest version of that configuration, as in ADC [1].

8. **System modeling**: abstracts the notion of a configuration from an instance of it and by fully describing the configuration, assists tools in maintaining the configuration's integrity, as in Jasmine [20].

9. **Subsystem**: provide a means to limit the effect of changes and recompilation, and for the environment to check the validity of configurations, as in Rational [27].

10. **Object pool**: optimizes the need for regenerating objects and maximizes the amount of sharing of derived objects, as in DSEE [14].

11. **Attribution**: permits the description of a system at a higher level of abstraction via its characteristics, as in Adele [2], rather than in terms of a composition of files from a lengthy file list.

12. **Consistency maintenance**: enables the environment to identify any inconsistencies and to preserve consistencies in creating and reusing configurations, as in CMA [10].

13. **Workspace**: provides isolation of work between programmers and distinguishes between a global, long-term repository for immutable objects and a private, shorter term repository for mutable objects, as in shape [34].

14. **Transparent view**: gives a viewing mechanism for a configuration from the main repository into a workspace with protection against unauthorized access, as in SMS [36].

15. **Transaction**: synchronizes and coordinates teams of engineers changing the same or different configuration, as in NSE [22].

These concepts provide a vocabulary that enables people to discuss automated CM sup-

port. No single CM system provides all these concepts. It is also possible to categorize [15] some CM tools suited to developers based on the sets of concepts they provide. The four categories are:

1. **The Check Out/Check In paradigm** is based on the repository concept. An example system is RCS.

2. **The Composition paradigm** is based on the repository and system modeling concepts. An example system is DSEE.

3. **The Change set paradigm** is based upon the repository and change set concepts. An example system is ADC.

4. **The Transaction paradigm** is based upon the repository, workspace, and transaction concepts. An example system is NSE.

Most developer CM systems are actually combinations of various paradigms, and each of these models implies a particular user process model.

As organizations begin to more formally define their process models and evaluate their process maturity levels, e.g., based on the SEI process maturity levels [11], it becomes clear that CM capabilities play a major part in attaining a higher process maturity level. Environments need to have CM integrated with their software engineering functions in order to support particular processes. Third-party CM tools and CASE tools generally avoid the integration problem so end users have to confront it. They need to know what integration issues are of concern and where CM fits in the integrated environment. There appears to be several mechanisms for addressing the tool integration problem:

- Buy an integrated project support environment (IPSE) [18], such as one based on the international standard, the portable common tool environment (PCTE) [37].

- Buy a tool coalition set [18] from CASE tool vendors, where the vendors do all the source-code level integration.

- Buy a meta-tool [38] that enables customers to rapidly develop their own highly customized set of tools.

Whatever mechanism is chosen, considerable thought needs to be given to the part that CM plays. For example, for every "save" command enacted in the editor, will a new version of a file be created? That is, how closely integrated does the editor need to be with the CM capabilities? For integration of CM functions, the main issues tend to relate to: who is in control—the CM system or some other tool; where is the data—in the CM repository or a tool's data dictionary; and what concepts are being integrated—will the auditing services work on data that is not kept in the CM repository. The integration problem is generally exacerbated because of a lack of an understanding of one's CM process.

3.2. Complexity of the CM Solution

It has become apparent that solving the CM needs for large organizations with long-lived software applications is complex. This is because the nature of the CM problem is complex for very large applications with many people spread throughout different physical sites. Many organizations spend most of their time working with existing applications and changing and enhancing them. The technology available to support automated CM is limited for large applications and limited to specific platforms, so many of the bigger organizations build their own CM systems. These organizations soon find out that their CM solutions are unwieldy and costly, and that maintaining the CM system is almost as burdensome as maintaining their own applications and supporting customers' applications. Because of these problems, they are now seeking a new CM system as the panacea for their CM needs. Whether it is updating their existing in-house CM system, starting from scratch by buying a third-party vendor one, or rebuilding their in-house CM system using available third-party tools as building blocks, finding an off-the-shelf CM solution to complex CM problems is not easy.

The lack of suitable CM technology to solve difficult CM problems also stems from the fact that the CM solution is very much dependent on solutions to other software engineering problems. For instance, the complexity of many CM problems would be reduced if we knew how to better describe the architecture of our software applications and families of applications in such a way that a CM system could easily allow programmers to make changes and propagate them, delete parts, and rearrange the software architecture to easily and safely accommodate and propagate change.

Consider the problem of maintenance in general. It is very difficult to maintain a large application over a long period of time. Many aspects change over time, including hardware technology, such as the change from time-sharing facilities to distributed, heterogeneous workstations; and tools, such as the change from simple compilers to structure editors. Providing for change is difficult, as is predicting what changes the future holds.

Three types of maintenance [3] are recognized these days:

1. **Corrective maintenance** is change involving bug fixes that are either scheduled to be made or are emergency patches, such as incorrect algorithms.

2. **Adaptive maintenance** is change that is required because some enhancement is needed or an underlying assumption has changed, such as the addition of new hardware.

3. **Perfective maintenance** is change that is required to make the application more amenable and robust to change for the future, such as making the software more modular, since it is known that change will occur.

CM technology of the past and present attempts to address corrective and adaptive main-

tenance through notions such as change requests, change control boards, and life-cycle state transitions. Assistance for perfective maintenance will come with improvements in software engineering techniques. Looking beyond actual CM concepts we see that improvements in software engineering techniques have enabled better CM concepts. For instance, techniques for structuring systems, such as module interconnection languages and higher level programming languages, have enabled programmers to better structure their applications by partitioning them into manageable portions. Thus, change can be isolated to certain portions with minimal effects on other portions and a better understanding of the effects of change to the whole system in general, such as a complete recompilation. Out of these improvements in partitioning, it was possible to design support in CM systems for such concepts as system modeling and derived object management.

CM is no longer a private problem within an organization. It became very public with the advent of third-party CASE and CM tools. The CM solution no longer involves just a simple decision of whether to develop an in-house CM system. It is now necessary to take into account all the available technology: whether a third-party CASE CM tool will actually meet the customer's needs and evolve as the customer's needs evolve, how the customer's software application will evolve, whether the platforms will change, how to deal with existing application software so that the CM solution is upward compatible, and so on. While these decisions still have to be made for a homegrown CM system within an organization, there are now more variables to take into account.

There are various factors that drive organizations to using automated CM support: government regulations, availability of CM tools, and recognition of the need for CM or improvements in existing CM support. Thus, the CM solution must take into account:

- **The organization, its specific projects, and individuals**: the enterprise's structure and politics. For instance, should there be a separate CM group or should each individual do their own CM?

- **Corporate CM versus Project CM versus Programmer CM**: the CM needs for the corporation, various projects, and for individuals. Should there be the same CM system for the whole organization, all projects, and all individuals?

- **The process**: defining and matching the CM process with the life-cycle process; implementing, enforcing, and monitoring the process.

- **Upward compatibility**: the solution should be upwardly compatible with existing software applications and the development/maintenance environment.

- **People**: their biases toward new CM systems and reluctance to use CM must be addressed, as well as training new people.

- **Roles**: each user role (such as programmer, tester, CM manager, project manager, and quality assurance manager) has certain views, duties, goals, and responsibilities. To what degree are these addressed in the CM solution?

- **Technology adequacy**: no "silver bullet" exists. That is, no single third-party CM system will immediately solve a customer‘s problems. The system requires customization and users need training.

- **Change accommodation**: need to easily accommodate for change, such as in technology (platforms, tools) and in process support to enable adapting to new policies, domains, and people.

- **Complex applications**: factors that make applications themselves very complex to develop and maintain. For instance, some factors pertain to size of application, distribution, heterogeneousness, international aspects, unknown application concepts, inadequate tools, and applications spread across different platforms and different media.

- **Buy versus build decision making**: whether to buy third-party tools, to build in-house starting from scratch, or to use existing building blocks. Which is the most cost-effective solution?

- **Integration and database**: integrating with other tools in the environment and determining whether one centralized repository or many repositories will be used.

- **Control level**: how much control will the CM system have over users? Will it have insignificant control where the CM system merely logs commands, or significant control where the CM system prohibits certain users from carrying out functions.

- **Automation level**: allowing that some aspects of CM will be done manually and some automatically.

Thus, the CM solution is as complex as its problem. As well as addressing the above issues, many preconditions need to be met (such as better system architecture descriptive abilities) in order to find suitable, long-term CM solutions. These preconditions involve finding solutions to general software engineering problems.

In summary, the present has brought about many CM systems, a better understanding of CM in terms of a vocabulary for concepts in CM systems, and a means of categorizing some CM support for programmers. With the plethora of tools comes the need to better understand how to integrate these into the environment. This requires a better understanding of the CM process. While many concepts are automated in CM systems, no single system provides all the concepts to suit all the needs of customers.

4. The Future

The future is characterized by five main issues: technology, process orientation, management, politics, and standardization. For technology, new requirements and better implementations are needed. For process orientation, CM process definition and automated support for implementing these are required. Management needs assistance in decision making about CM systems, such as whether to buy or build decision a CM system. Politically, it appears that the U.S. government may eventually require its contractors to have a certain level of CM support in their environment. For standardization, CM is being recognized as a key factor in environment framework standards.

The SEI is attempting to address the technological, process-oriented, managerial, political, and standardization issues through its CM services model work. This is our vision for a technical solution and it has implications on each of the five main issues.

4.1. Challenges

The main challenges confronting us in the future can be categorized as follows: technological, process-oriented, managerial, political, and standardization. Each is briefly described below.

4.1.1. Technological Issues

Some CM tool vendors recognize that the CM solution for customers is only about 10% technological—the rest is improved management, process, and user training. This 10% figure applies to existing technology. For the future, new CM functionality requirements need to be implemented, along with addressing tool integration with CM functions and better support for software architectures [35]. These include:

- **Switching CM Capabilities**: turning off/on various features in CM systems statically or dynamically, thereby customizing the CM system. This could result in various levels of control that may suit different phases of the life cycle of the application or different projects or people.

- **Interoperability between CM Systems**: providing for interoperation between CM systems. Large organizations generally have groups using different CM systems because these run on different hardware. For large system integration of software, it would be ideal after system integration to keep and continue to maintain the CM information from the various origins. A way to do this is via interoperability. For instance, all the CM information from various systems could be combined under one system. Some systems, such as ADC, already provide for the importation of SCCS files into the ADC repository.

- **Global Perspective on CM Repositories**: most existing CM tools allow the user to choose the scope of the repository; for example, a repository per directory of files or a repository per set of directories. Since many repositories require uniqueness in file naming, users generally choose a repository per direc-

tory. Thus, a project can result in an abundance of directories. For a user to access the CM information in these directories, it is necessary to individually access each directory—there is no higher level view into all the repositories. Thus, a global perspective would allow a user to view all repositories as one global repository without having to individually traverse directories.

- **Distributed CM**: with distributed, heterogeneous environments comes the need to resolve distributed CM capabilities. Is the CM to be distributed or do customers only need distributed access to a central CM repository? In very large organizations, the change control process and change control board are the focal point for the CM system. Changes are either done locally or globally. There tends to be two levels of organizational CM facilities involving one host as the official CM repository, which really serves as a passive depository or archiving engine. The change management capabilities are used to control all access to that engine. While the build procedures cannot generally be controlled from that engine (since the builds have to be done on the remote targets), it is possible to send the derived objects to the CM engine along with its generation details. An issue for further thought is how much of the distributed CM problem can be solved by using a distributed file system, as opposed to requiring a more innovative solution.

- **Perfective Maintenance Support**: supporting perfective maintenance allows users to restructure their software to make it more amenable to change. This generally involves reverse engineering, reengineering, and semantically based, change impact analysis capabilities.

As part of addressing tool integration, the SEI is attempting to merge the IPSE and tool coalition technologies (presented earlier in Section 3) into a federated architecture [18] that would encompass the benefits of both kinds of technologies. This architecture would specifically provide for integration of CM capabilities.

4.1.2. Process-Oriented Issues

Users need to better understand their CM processes so that they can demand better supportive implementations for them. This requires a detailed definition of CM processes; an understanding of how much control will be enforced compared to how much guidance will be given by the process manager; adequate implementations; and monitoring of how well the process is followed and where improvements can be made. Better understanding and implementations of processes enables improved support for users in attaining a higher quality of product, more time for being productive on creative tasks, and better forecasting of software costs.

For instance, the CCC [8] system supports a process. Certain steps must be carried out in a certain order, but there is little automated guidance as to which steps should be done when. The order of commands in the menu suggests the command order, but this is really a simple guide. At any point in time a user cannot immediately and automatically know the next step. Also, to implement a process, more than just step sequences (that is, control flow) are

needed; some semantic context is required. For instance, the CCC turnkey system keeps an audit trail of the CCC commands that the user issues. But, the audit trail for emergency fixes gives no indication whether any file was checked out and changed. So there is no data associated with the audit trail, only some logging of actions. This information may be insufficient for a particular organization—while a simple mechanism for an audit trail is provided, customers may want more semantic content in the audit trail. Thus, a process implementation involving control flow of commands and avoiding capturing of data state is likely to be insufficient for the customer.

4.1.3. Managerial Issues

To solve the CM problems in an organization, it is necessary to get better management buy-in; that is, give management an understanding of the complexity of the solution and hence the costs and tradeoffs. This enables a better commitment of resources to the solution, such as tools, people, and money.

As part of finding a solution, management will need to be able to evaluate CM systems and their capabilities. After that it is necessary to deal with the technology transition issues [23] of introducing CM technology into the workplace, such as convincing people to use CM. It is always necessary to customize the CM tool to suit the needs of the workplace and to understand how CM fits into, and affects, the existing environment. A CM tool can no longer be viewed in isolation from other aspects of software development since it must be integrated with other capabilities in the environment.

Management must be prepared to make the "buy versus build" decision in examining possible CM solutions. This requires an understanding of what the cost drivers are that help influence this decision. There is a continuum on which such a decision can be based. At one end is the "build in-house" decision with certain benefits, such as the customer having total control over the CM system but the tradeoff is that the organization must bear the brunt of the cost of the CM system. On the other end of the continuum is the "buy third-party" decision, with the benefit that the tool vendor bears the brunt of the development and maintenance of the CM system, but at the expense of the customer losing control over the functionality of the CM system. Management needs to make judgements based on quantitative and qualitative issues. Issues and decision-making drivers need to be recognized and quantified when possible to provide information for managers to make their decisions.

4.1.4. Political Issues

It appears likely that future government contractors in the U.S. will be required to use certain CM facilities in order to procure a contract. For example, a contractor would have to be a level 3 organization, where that level is based on the SEI capability maturity model [11]. This implies certain software engineering practices and kinds of tools would have to be used.

4.1.5. Standardization Issues

Standards work involving environments and CM has begun, such as the U.S. Navy Next Generation Computing Resources Project Support Environment Standards Working Group's efforts at defining a conceptual model [26] for addressing tool integration in a project support environment. Its result will be a services reference framework into which CM services can be placed.

Other efforts at solving CM problems are spread across many fields, such as the database and electronic computer-aided-design (CAD) fields with its CAD Framework Initiative standardization effort. It is likely that many similar efforts will be combined. This would speed up progress and enable a conceptual CM framework to be defined that encompasses all these domains.

4.2. A CM Services Model

One way that the SEI is starting to address some of the above issues is through the definition of a CM services model. A CM services model is a conceptual framework for a set of well-defined services. A "service" is meant to be a particular CM functionality. (It is further discussed below.) "Well-defined" means that a service is defined in such a way that its semantics, interface, and other properties are understood enough to be included in framework reference models and, with possibly different mechanisms, to be implemented.

The services in the model take into account the software engineering marketplace's need to apportion and distribute functionality. That is, CASE tools and environments provide parts of CM solutions. Customers buy these pieces as building blocks for their solution (since no single CM system is a panacea). The CM solution is generally spread across tools. For example, customers use SCCS and may change and enhance a *Make* facility to provide more of a system modeling capability and include some workspace facility. Thus, the services model, in essence, is intended to provide plug in/plug out, "black box" capabilities. The initial set of 50 includes services, such as repository, system modeling, version binding, derived object management, change management (normal and emergency fixes), workspace, transaction, change control board, change impact analysis, change boundary, merging and conflict resolution, version differentiation, project context, configuration identification, release identification, access control, change propagation, report generation, audit trail, status indication, CM planning, snapshotting, consistency management, statistics gathering, and notification.

The services cover a broad spectrum of requirements [12]. They represent a mixture of several viewpoints: end user, environment builder, and tool integrator. The end user wants to pick and choose capabilities for the CM system, the environment builder wants to provide tailoring features for the end user, and the tool integrator wants to mix and match existing capabilities and devise a cost-effective CM solution. For instance, the end user wants change management capability to control changes. The environment builder wants to incorporate traceability features so that the end user can keep track of configuration items during

their life cycles. The tool integrator wants to integrate an existing version control system (such as RCS) with other CM capabilities from another tool (such as the NSE transaction and the ADC change set).

While the services model is a technical contribution, it does have implications on other aspects:

- For technological issues, the model suits a client/server approach for tool integration [6].

- For process-oriented issues, services can be grouped and tailored to meet specific process requirements, such as those for specific user roles (whether it be a programmer, a project manager, a configuration manager, a tester, or a quality assurance manager).

- For managerial issues, the cost of the service can be determined and hence, provide better forecasting of CM expenses for management.

- For political and standardization issues, the services can be standardized and well-defined, enabling environment framework designers and standards committees to more easily discuss CM and place CM in their models.

The choice of services in the model comes from various sources. These sources include surveying CM systems [12] to determine what concepts exist and are thus implementable; examining how CM systems combine sets of concepts [15] to address certain kinds of CM needs; evaluating how to use various CM systems [27, 19, 22]; examining CM marketing literature for user requirements; recognizing what roles need to be supported by CM systems; identifying CM practices [11] that need to be supported; and analyzing user experiences in building and using customized, in-house CM systems.

The services model ties together process models and user roles. It does this through the definition of services independent of user role and process. The customer picks and tailors the services to suit a particular role for a particular process. Figure 4-1 gives an indication of how the services model is to be used. The services are used by picking a set of services specific to the customer's needs. These services will be made out of various mechanisms that have been customized and integrated together to meet certain policies.

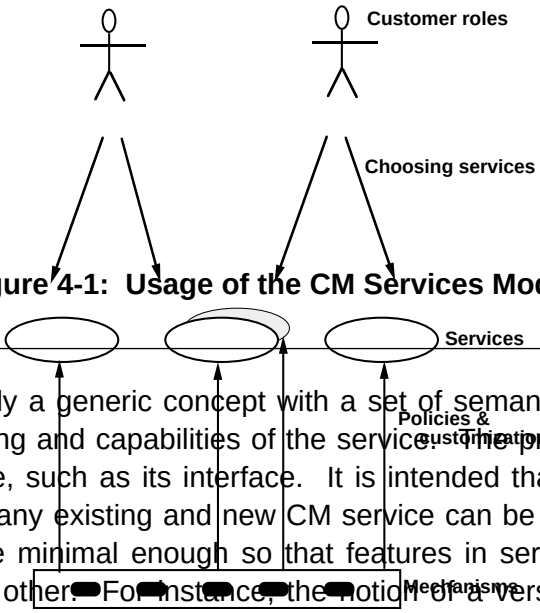

Figure 4-1: Usage of the CM Services Model

A service is fundamentally a generic concept with a set of semantics and properties. The semantics are the meaning and capabilities of the service. The properties are the items of interest about the service, such as its interface. It is intended that the services model be "open" in the sense that any existing and new CM service can be defined within the model and that each service be minimal enough so that features in services do not overlap but rather complement each other. For instance, the notion of a version tree is considered a property of a repository service whose semantics involve providing relationships between items. The version tree is not a distinct service though, whereas the repository is.

The main difficulty in defining the services is developing the appropriate criteria that enables the choosing of the services. To accommodate the intention of openness and evolution for the services model, the main criteria needs to be the following: a CM functionality is a service if it is a replaceable unit, that is, if it can be treated as a plug-in/plug-out entity. To complete the definition of the model, other issues need to be resolved. These include the need to distinguish between a service specification and instance so that the definition and the usage of a service can be separated; find a boundary between what is a service versus what is a mechanism that implements that service; define the interfaces, tailoring capabilities, relationships, and effects of the services; complete the list of services, their semantics, and properties; categorize the list based on various aspects such as functionality, role, requirements, or effect; determine a decomposition strategy that captures

layers of abstraction and granularity of services; decide whether services should be allowed to have overlapping capabilities; specify (at an abstract level) the nature of the implementation mechanisms; use the model to define various process models and policies for various users; understand how to implement and integrate services as proof of concept; formally specify the services; and define existing CM systems in terms of the services.

How useful is the services model for the present and the future? The answers depend on the kind of "user." For CASE tool vendors, a CM services model gives them a vocabulary for uniformly describing the functionality of their CM system or the CM capabilities within their tools. For customers of CM systems, it provides a check list of functionality for evaluating and choosing CM capabilities and for matching their process to specific functionality. For tool integrators, it provides a means to discuss integration interfaces. For CM system developers, the model can provide a specification of which mechanisms need to be in place in order to implement certain functionality. For environment standards developers, it provides a partial reference model, and for government organizations it provides a basis for verifying and analyzing CM capabilities.

5. Conclusion

CM capabilities are the foundation of any software development environment. Good CM support makes for a good environment. Bad CM support makes an environment unusable. A CM solution represents a microcosm of all issues affecting an environment including technology transition, user requirements, roles, integration, databases, mechanisms, CM technology, process support, education and training of users, and managerial and organizational decisions.

This paper has presented where we were, where we are, and where we plan to go in order to solve our CM needs and problems. The past involved homegrown CM systems. The present consists of a plethora of third-party tools and a better understanding of CM technology and the depth of complexity to a long-term, CM solution. The future holds many avenues for progress in addressing technology, such as new functionality; process, such as better process support involving semantics of data; management, such as improved recognition of the need for resources; and politics and standards, such as standardizing CM services. A CM services model was introduced as a way of starting to address many of these issues. In general, the ramifications of introducing CM and the cost of using and doing CM are becoming understood. There has been a lot of progress with CM and this will continue.

One way of viewing many of the issues raised in this paper is to view them as four philosophical questions that examine CM in relation to software engineering.

1. **What is the CM boundary?** It is not clear what the dividing line is between what is CM and what is not CM. For instance, a customer has a requirement for traceability of all the application's artifacts, including every version of an artifact. Is this a CM requirement or something else?

2. **When is a problem not a CM problem?** Depending on one's perspective, it is not always clear whether a problem is a CM one. For example, I have a slide set for a presentation. I make another slide set for a second presentation using slides from the original set. I invoke my editor and cut and paste some slides from the first presentation. Is the creation of this second slides set a CM problem since a variant is created or is it a reuse problem since a copy is made? Is it a software architecture problem in finding a suitable structure and choosing the appropriate granularity of slides or is it a database problem since a derivation relationship exists?

3. **When is a solution not a CM solution?** Given a CM problem, it can be difficult knowing whether the solution is a CM one. For instance, when surveying the concepts in CM systems, what concepts are specific to CM and what concepts are not? Some seem to encroach on or come from other fields such as databases (as with the transaction and attribution concepts) or process modeling (as with the life-cycle models that are the crux of the CM system).

4. **What part of the software engineering problem is CM?** It is not clear whether general software engineering issues, such as team support, process

support, and tool integration are areas that are part of CM. Yet CM vendors generally need to address these issues when giving the customer an off-the-shelf solution. This question also arises under the guise of "Where do we put CM?" when committees are trying to decide upon the architecture for their environment framework reference models.

These questions will be continually debated. In the meantime, CM vendors and environment builders are developing CM capabilities and organizations are forming their CM solutions. Thus, both are providing us with more information on how to answer the above questions. At the moment, a long-term CM solution that supports long-lived, changing software for large organizations with diverse software applications will not be found in a single, off-the-shelf CM tool or environment. Other aspects of software engineering need to be addressed in unison, such as process modeling, software architectures, team support, and tool integration. CM has an impact on these aspects and vice versa. CM is a keystone to the software engineering problem and it should not be viewed in isolation from other software engineering problems and solutions.

6. Acknowledgements

I would like to thank Peter Feiler for supporting me in writing this paper and his invaluable insights, as well as the following people for their review and advice: Nadine Bounds, Alan Brown, Alan Christie, Gibbie Hart, Fred Long, Carol Sledge, Howard Slomer, and Paul Zarrella. A special thanks, too, to the IFIP Congress reviewers, Sandy Brenner, and the SEI technical writers.

References

1. Software Maintenance & Development Systems, Inc. *Aide-De-Camp Software Management System, Product Overview.* Concord, Mass., 1989.

2. Estublier, J. A Configuration Manager: The Adele Data Base of Programs. Proceedings of the Workshop on Software Engineering Environments for Programming-in-the-Large, June 1985, pp. 140-147.

3. Arthur, L. J. *Software Evolution: The Software Maintenance Challenge.* John Wiley and Sons, US, 1988.

4. Babich, W. *Software Configuration Management.* Addison-Wesley Reading Mass., 1986.

5. Bersoff, E. H. Bersoff, Henderson, V. D., and Siegel, S. G. *Software Configuration Management.* Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

6. Cagan, M. R. "The HP SoftBench Environment: An Architecture for a New Generation of Software Tools". *Hewlett-Packard Journal* (June 1990), 36-47.

7. CASE Consulting Group, Inc. "Configuration Management". *CASE Outlook 90* , 2 (1990).

8. Softool. *CCC: Change and Configuration Control Environment. A Functional Overview.* 1987.

9. Schmidt, E. E. *Controlling Large Software Development In a Distributed Environment.* Palo Alto Research Center, December 1982.

10. Ploedereder, E. and Fergany, A. A Configuration Management Assistant. Proceedings of the Second International Workshop on Software Version and Configuration Control, ACM, US, October 1989, pp. 5-14.

11. Paul, Mark et al. Capability Maturity Model for Software. Tech. Rept. CMU/SEI-91-TR-24, ADA240603, Software Engineering Institute, Carnegie Mellon University, August 1991.

12. Dart, Susan A. Concepts in Configuration Management Systems. Proceedings of the 3rd International Workshop on Software Configuration Management, June 1991, pp. 1-18.

13. Deitz, D. "Pulling the Data Together". *Mechanical Engineering* , (February 1990).

14. Leblang, D. and McLean, G. Configuration Management for Large-Scale Software Development Efforts. GTE Workshop on Software Engineering Environments for Programming in the Large, June 1985, pp. 122-127.

15. Feiler, Peter H. Configuration Management Models in Commercial Environments. Tech. Rept. CMU/SEI-91-TR-7, ADA235782, Software Engineering Institute, Carnegie Mellon University, March 1991.

16. Przybylinski, Stanley M., Fowler, Priscilla J., and Maher, John H. *Tutorial presented at the 13th International Conference on Software Engineering, Austin Texas.* Software Engineering Institute, Carnegie Mellon University, US, May 1991.

17. Habermann, A. N. and Perry, D. E. Well-Formed System Compositions. Tech. Rept. CMU-CS-80-117, Dept of Computer Science Carnegie Mellon University, March 1980.

18. Brown, Alan W., Feiler, Peter H., and Wallnau, Kurt C. Understanding Integration in a Software Development Environment. Tech. Rept. CMU/SEI-91-TR-31, ADA248119, Software Engineering Institute, Carnegie Mellon University, December 1991.

19. Graham, M. and Miller, D. ISTAR Evaluation. Tech. Rept. CMU/SEI-88-TR-3, ADA206429, Software Engineering Institute, Carnegie Mellon University, July 1988.

20. Marzullo, K. and Wiebe, D. Jasmine: A Software System Modelling Facility. Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, December 1986, pp. 121-130.

21. Feldman, S. I. Evolution of MAKE. Proceedings of the International Workshop on Software Version and Configuration Control, Siemens, Germany, January 1988, pp. 413-416.

22. Feiler, Peter H. and Downey, Grace. Transaction-Oriented Configuration Management. Tech. Rept. CMU/SEI-90-TR-23, ADA235510, Software Engineering Institute, Carnegie Mellon University, November 1990.

23. Przybylinski, Stanley M., Fowler, Priscilla J., and Maher, John H. Tutorial: Software Technology Transition. Proceedings of the 13th International Conference on Software Engineering, USA, May 1991, pp. 105.

24. Johnson, W. Bringing Design Management to the Open Environment. High Performance Systems, June 1989, pp. 66-70.

25. Brown, Alan W. and Feiler, Peter H. The Conceptual Basis for a Project Support Environment Reference Model. Tech. Rept. CMU/SEI-92-TR-2,, Software Engineering Institute, Carnegie Mellon University, January 1992.

26. Next Generation Computer Resources. *Reference Model for Project Support Environment Standards,* U.S. Navy, April 1992.

27. Feiler, Peter H., Dart, Susan A., and Downey, Grace. Evaluation of the Rational Environment. Tech. Rept. CMU/SEI-88-TR-15, ADA198934, Software Engineering Institute, Carnegie Mellon University, July 1988.

28. Tichy, W. Design, Implementation and Evaluation of a Revision Control System. 6th International Conference on Software Engineering Tokyo, September 1982, pp. 58-67.

29. Glasser, A. L. The Evolution of a Source Code Control System. Proceeding Software Quality Assurance Workshop, Sigsoft, New York, NY, November 1978, pp. 1-4.

30. *IEEE Guide to Software Configuration Management.* 1987. IEEE/ANSI Standard 1042-1987.

31. German Chapter ACM, GI, Siemens AG. , Grassau, W. Germany, January 1988.

32. ACM SIGSOFT, IEEE CS, GI. , Princeton, NJ, November 1989. Software Engineering Notes, Vol 17, No 7..

33. ACM SIGSOFT, IEEE CS, GI. , Trondheim, Norway, January 1991.

34. Mahler, A. and Lampen, A. Shape—A Software Configuration Management Tool. Proceedings of the International Workshop on Software Version and Configuration Control, Siemens Germany, January 1988, pp. 228-243.

35. Shaw, M. Toward higher-level abstractions for software systems. In Data and Knowledge Engineering 5, Ed., *Data & Knowledge Engineering*, Elsevier Science Publishers (North Holland), May 1990, pp. 119-128.

36. Cohen, E., Soni, D., Gleucker, R., Haslin, W., Schwanke, R. and Wagner, M. Version Management in Gypsy. Proceedings of the ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, November 1988, pp. 210-215.

37. Boudier, G., Gallo, T., Minot, R., and Thomas, I. An Overview of PCTE and PCTE+. Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Engineering Environments, Boston, MA, , USA, December 1988, pp. 248-257.

38. Bloor, R. "The Software Tools' Software Tool". *DECUSER* , (April 1990).

39. Whitgift, D. *Software Configuration Management: Methods and Tools.* John Wiley and Sons, England, July 1991.

Table of Contents

List of Figures