Software Engineering Institute

Technical Report

ESD-TR-86-209

CMU/SEI-86-TR-5

December 1986

Summary of the SEI Workshop on Software Configuration Management

Katherine E. Harvey

Carnegie Mellon University Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler
SEI Joint Program Office

Table of Contents

Summary of the SEI Workshop on Software Configuration Management

Abstract

1. Participants

Bradley Brown	Boeing Military Airplane
James Collofello	Arizona State University
Robert Glass	Seattle University
Ted Keller	IBM Fed'l Systems Div.
Richard Parten	Lockheed
Mary Shaw	SEI
Howard W. Tindall	Martin-Marietta
James E. Tomayko	SEI (host)

2. Introduction

Following is a summary of the discussion held during the Software Configuration Management meeting at the Software Engineering Institute in Pittsburgh on July 16, 1986. In this document I have tried to determine the major concerns brought up and the conclusions reached during the day tong discussion. The discussion ran in many directions, often changing topics quickly and not returning to the original subject for quite some time. Therefore, I did not try to summarize the discussion chronologically, since I felt that would be more confusing than informative. I have, instead, tried to sort the various concerns and conclusions into specific areas and have summarized the discussion of each major point brought up in those areas.

3. Overview

3.1. Definitions of Software Configuration Management

The basic definition of Configuration Management that the workshop participants more or less agreed upon seems the best place to start, since the definition is fundamental to the discussion of Software Configuration Management (SCM). Jim Tomayko, the host of the workshop, put as his capsule description of Configuration Management: "the disciplines and techniques of initiating, evaluating, and controlling change of software products during and after the development process." This definition met with general approval, although the discussion as a whole brought out a much more complex and discerning description. The most apparent concept missing from the original definition was that SCM is a fundamental and essential management tool for successful software development projects. It is more a management concept than a concrete structure and is invaluable to the organized and rapid output of a software product.

Although the concept of SCM was thought to be fundamental to the maintenance of software products, the workshop members believe that associating SCM with maintenance is misleading. Configuration management should not start simply when a software product reaches the maintenance phase; the whole development process must be managed in such a way that SCM can work properly. For instance, if the original designer does not document the work properly, then the configuration management process breaks down, because later changes create problems not immediately apparent based on the existing documentation. SCM, therefore, is an integral part of the entire software design and development process and a vital part of all software engineering.

3.2. The state of Software Configuration Management today

One of the major points of discussion was how Software Configuration Management is being used by the software engineering community today. No one at the meeting thinks that SCM is being used effectively as a management tool; in fact, just the opposite. Although there have been many corporations with solid SCM programs, many others produce software today with either no program whatsoever or programs that hinder rather than help. What is wrong with the SCM programs today?

A major problem is the lack of a widespread understanding of the usefulness of a solid SCM program. Although many large companies do have configuration management systems, often when they turn a project over to a smaller contractor, the Software Configuration Management is left up to that contractor, who often chooses to do nothing. If the configuration management is bad, one can almost guarantee that the documentation will be bad. Then, when the development process for a software product is over and it goes into maintenance mode, the contractor turns over a software project with incomplete documentation. So the company left with the project is lost, they start playing around with it, and they are left with “spaghetti” software. According to the members of the workshop, this happens all the time, even though many of these projects are expected to interact with others.

One key factor in an effective configuration management system is a solid Configuration Control Board structure. However, in most companies today the importance of the boards and their members is overlooked. Often the people put on these boards do not have the training or experience to make decisions about changes or problems in software products. One example is an entire Software Configuration Management division that is virtually a hindrance to the organization. In this organization, when a change request is written (often only a paragraph or less of information) and sent into headquarters it goes to the Configuration Control Division (CCD). However, this division’s job is simply to put a number on the change report (CR) and send it out, without any kind of board meeting or discussion whatsoever. This CR is sent out for review to people who can’t possibly tell from a couple of sentences of information whether the change is a good idea. Afler several weeks, the reviewers send the request back to the CCD with “nonconcur" or “concur” stamped on it, and often it takes months before any real action is taken on the document. If the change is approved, it goes to the implementation organization, which writes the functional specification and the detail design with no review of either document. This same organization does all of the coding and testing, without ever consulting a review board or the originator of the request; then when the change finally shows up in the field, the originator probably won’t even recognize it.

Many times the people at higher levels of large projects don’t understand software and think of it as simply "another subsystem." It becomes a difficult task to convince these project managers that on a large complex system, or any project that has as its root a data system, the software is an integral function of the project. Often in these projects the usefulness of a configuration management system is overlooked and therefore this vital management tool is not used properly. Many times attempts are made to use other methods, like the CSSR or C Spec. system, to maintain control over projects. But these programs tend to be cursory measurements of progress and costs without ever getting down to the real work of change management. It is in the configuration boards that changes are discussed and information gets moved around, the place where sleeves are rolled up and the nuts and bolts of the software are laid out.

So it is apparent that there is a great need for improving Software Configuration Management. The problems are large and widespread; of course, they won't be solved overnight. However, the workshop participants had a great many ideas about the components of an ideal configuration management system. These may provide the base for educating future software engineers to better manage their projects through Software Configuration Management.

4. The Software Configuration Management System

4.1. A general picture

The participants more or less agreed that there are too many unpredictable circumstances in the corporate world to build a generic all-purpose configuration management structure. However, it is possible to sketch in certain key elements without creating a definitive structure. Some of the elements are just fundamental characteristics of a SCM system, while others are more subtle details that will create a more efficient management machine.

All large system projects have systems of Change/Configuration Control Boards (hereafter known as CCB). The structure of the CCB may differ widely according to the type of project, the company in charge, and the size or cost of the program. However, the CCB are a necessary element to almost any SCM structure. These boards work in a kind of waterfall structure with a great number of control boards at the lower levels which feed up into the higher levels. Change usually bubbles up from the bottom where the programming activity is boiling. At times there is reverse traffic when change requests come down from either the customer or the system manager, but the vast majority come from the area of the most programming activity. Therefore the greatest number of boards is at this bottom level of great activity, and these boards should be the most active.

4.2. Ideal CCB characteristics

One of the most important characteristics for any control board, but especially the lower level boards, is that they should be active. Because this is where information is passed around, where you begin to see the project's shape and direction, it is vital that the boards be a well-used and functioning body in the SCM structure. The CCB should be a place of discussion, where any problems or requests that come up in a project get hammered out. On large projects these board meetings often last longer than a full day, but the work being done in them is vital.

Because this work is so vital to a project, "casual involvement" simply cannot exist in the CCB system. It is important that the management people on each board look into every CWDR that comes before them. Even if the change or bug is presented in a very casual or non mission-critical way, it is the duty of the board members to look into it as if it were. If board members allow the casual nature of a presentation to affect their decisions and evaluations, problems may be overlooked that could escalate later into emergency situations.

There should be a route for emergency changes so that the system won't break down during emergency situations. There should also be a CCB appeal route. This means that it would be possible to go to a higher CCB if the originator of the change request deemed it absolutely necessary to reverse the original board's decision. This would help keep the board meetings from becoming "shouting matches," and help people discuss things rationally. However, the appeal route must be carefully controlled (perhaps by upper level boards making decisions as to which appeal request should be accepted) in order to keep the authority of the lower boards intact.

It is important not to limit the number of boards because of past SCM practices or "efficiency". It is actually more efficient to have as many boards as possible within cost and common sense parameters. Each board should have the minimum number of people possible needed to make decisions. Therefore each CCB's jurisdiction should be well documented, and only those people directly involved in or affected by changes in their jurisdiction need to be at the CCB meetings. This way, only vital people are involved in their particular CCB decisions, and other important people who not directly involved don't waste their time.

4.3. Ensuring proper documentation

At a basic level, CCB should be involved with all changes taking place at the project level, using a lot of discussion and review for each change being made. For this to happen, documentation standards throughout the development phase of a system must be enforced. In every SCM structure it would be a good idea to have a division to make sure that the original developers are writing down everything. Documentation rarely gets done without outside influence by those developing the code, and almost never gets done post-facto (certainly not accurately). If there is no documentation, there is nothing to control. Documentation "enforcers" are a good idea for a strong SCM system, provided their authority is well documented and strictly monitored.

These various characteristics of a good SCM structure may vary a great deal, especially when the existing authority levels are different. The authority hierarchy in a company or program has a great deal to do with the configuration management system, and all the elements that have been talked about so far rest upon well-organized authority levels.

5. Authority in SCM Systems

5.1. Authority hierarchies

Although, as previously stated, the CCB areas for discussion, the final decision-making authority should lie with one individual. The control boards do not vote on changes; one person makes a decision under advisement. This is extremely important when trying to avoid interproject politics and to keep a program oriented toward its proper goal. The higher level boards have greater authority, of course, than the lower boards, and the system level CCB belongs at the top of the pyramid. It is the head of the system level CCB who has authority to make the final judgments on CR/DR and any last minute emergency "patches," although this authority is usually delegated to lower level boards who are more often confronted with the problems as they come up. This means that whoever is making those final decisions had better be pretty sharp, or the program is headed for trouble.

It is also healthy to a project to have a slightly adversary relationship between the software design manager and the head of the program. The design manager fights for the needs of a particular area, while the head of the program sees a more overall picture, hardware systems as well as software. If both these people are well trained in project management, then the adversary relationship will provide much needed checks and balances within the project system.

The various CCB should have documentation readily available to them detailing specific areas over which they have authority. Each board needs to be sure what decisions they have authority over and how much authority they have to make a decision. When CR/DR come up, there should never be confusion as to who is responsible for looking into them. So it is very important that CCB jurisdiction and authority cover every area at some level, especially those critical to the project, and this authority must be documented. For example, if the Testing and Evaluation division discovers a DR, it must be clear whether they have the authority to make changes in the program or if they need the authority of a higher board to make the change, and whether this authority changes in the event of a mission critical DR. At the workshop, two experiences were given as examples: in one situation the Test/Evaluation people did have authority to make changes even on non mission-critical DR, while in the other situation they simply reported the DR to higher boards for action, or the Evaluation people simply figured out ways to work around non mission-critical DR. The responsibility for these decisions need to be well documented to avoid confusion.

5.2. Key Authority concepts

It is very important to understand that authority levels cannot be generically structured to fit any situation. Usually the structure of any given SCM system depends on the authority levels already in existence in the particular company or program involved. Any project manager coming into a company or program must have a good appreciation for the existing authority hierarchy. The Software Configuration Management system must be molded around those authority levels.

Oftentimes the way people perceive problems can create difficulties. While one person may see something as a “problem,” someone else may see it as simply a “change.’ Who has the authority to deal with these varying perceptions? It may be that it comes under the authority of each CCB head, or that an entirely different division or CCB should be set up to deal with this question. Once again, this will probably depend on the already existing authority hierarchy. However, it may also depend on the people in the program, the size of the project, and various other management considerations.

It is also important for each company that goes under contract with another to have appreciation for the existing authority hierarchy in the other company or organization. When everyone involved in a project understands the authority structure and the way they are expected to work within it, a smoother operation and a more productive work atmosphere will result.

6. Tools for Software Configuration Management

6.1. Tools of the trade

Quite a few methods for maintaining control over change were discussed. Many were technical devices that are well documented and available, so the group spent very little time on these. Others were not discussed necessarily as “tools,” but I think that they could be labeled as specific tools for software configuration management and that this would be a good place to summarize them. First, let’s look at the naming and/or numbering of products.

One of the most important aspects of any system for naming software products is that the name be specific as well as unambiguous. A specific example was brought up regarding NASA back when the name of a software system matched the mission for which it was being built. This, however, soon became a problem. In these projects there is usually a long time between starting to build a software system and the time the mission it is intended for finally flies. Often halfway through the maintenance life cycle of this software, major changes are made in the project: payloads may be swapped or scrapped, as may the mission vehicles, and so on. When these changes are made, the name of the mission is often changed. Then, one is left with a software system named for a mission that may not fly for years if it ever goes up at all. It is easy to see how this could become confusing. Therefore, the software is now named and numbered in a completely separate way so that there can be no relevance to the mission for which it’s being developed. What is important to see in this example is that the naming system had to be adjusted to become more specific to the product as well as less ambivalent.

Because various divisions may have different names for a single system, and because communication must eventually extend beyond the purely technical community, it is important to be able to see how the nomenclature evolves and how the various nomenclatures relate to one another. IBM uses a waterfall diagram to show the path of each particular system and its various names along the way. But perhaps more importantly, IBM puts on the same page a cross reference list. Since each nomenclature may for a particular software build have three different names with each of these names understood by different divisions, the cross reference list is important for clear understanding and communication. Using diagrams is also a useful tool for communicating with those outside the technical community.

A management tool that might not be distinctly thought of as a SCM “tool” is that of using “freeze dates” when putting out incremental releases of a software system. The example at the meeting went something like this: Usually, the top management people on a project are anxious to see some sort of working software even though the software designers are still working out the bugs in a code and may be reluctant to release it. In a case like this, having freeze dates for the software to be turned in will force the developers to release what they have even if they feel it is incomplete. Usually the first release will be chaotic but this will give a good idea about where to go and what needs to be fixed, and the consumer has a working product. Even if it has a lot of DR, having a completed product is a positive incentive and will improve the working atmosphere. The freeze dates must be rigid; if the developers don’t get their projects in on time, they won’t be included in the release. If this isn’t enforced, the computer people will keep fiddling around and changing things, and the entire program will fall behind schedule. Once again it is important to remember that implementing a tool like this will depend a great deal on the existing situation.

7. Documentation and Credibility

7.1. Documentation

At one point in the day's conference, Jim Tomayko asked the group if they knew whether anyone paid attention to the standards for software configuration management put out by IEEE. No one at the meeting had even heard of them. They were aware of the Department of Defense Standard 2167, but it was generally acknowledged that this was overlooked by most program managers. The standards get overlooked because the rigorous documentation requirements that they establish are seen as cumbersome and so documentation standards do not get enforced. At first, ignoring the standards seems easier for both the managers and developers. It isn't until they are waste deep in the mire of unmet schedules, undocumented software with hundreds of unseen DR, rising costs, and consumers anxious to see this project that is now out of control, that the importance of enforced documentation standards is apparent. How do you motivate people to use cumbersome standards when they haven't been "burned" by past experience? Obviously, standards for software configuration management are a useful tool, but getting people to use them is anot her matter.

It cannot be said enough that without documentation there is nothing to put under configuration control. There must be a valid functional specification document in order to get past the Preliminary Design Review or there is nothing to put under the management system and you're already off schedule. A brief list of documentation includes:

- requirement specification documents
- functional specification documents,
- detail design documents,
- user manuals,
- maintenance manuals,
- interface control documents,
- memory layouts,
- test plans,
- and the code itself, of course.

All of this, plus more not mentioned here, comes under maintenance control, unless it is subject to a *project specific* waiver. Because many of these documents are scrapped when a product reaches maintenance phase, it would be best perhaps to maintain a configuration index for each product so that enough documentation is maintained for configuration control during the maintenance phase.

Even in the essential area of documentation there must be consideration for the project involved. If the project is large, the managers are usually more careful about enforcing documentation standards because the project as a whole is probably being approached in a very careful and cautious way. However, in a smaller project, SCM tends to take a back seat and the documentation, therefore, doesn't seem important or economical. Sometimes full rigor on the SCM and documentation standards can be relaxed slightly on a smaller project, but then you need someone in command who knows when full rigor can be relaxed and when it should be enforced. However, good documentation will always help configuration management people to make sounder judgments and more credible evaluations.

7.2. Credibility

Good basic documentation is the basis for Configuration Control Board evaluations on the issues before them. In order for CCB to make intelligent, rational, and credible decisions on CR/DR, certain data are necessary. These data should be well documented so that the CCB evaluation of the data will carry weight. This list of necessary data was developed at the workshop:

- The size of the change.
- Alternatives are there any? Would it be relatively simple to work around whatever is being changed?
- The complexity of the change. Does it reference other systems? Does this system support other software or rely on other support software that would need to be changed accordingly?
- The need date. Basically, the board needs to know how much time they would have to make the change and test it, before the consumer needs a working product.
- Impact. This is related to complexity. What kind of effect will this change have on subsequent work? The board needs to took down the road a bit and see where the project is going.
- Cost. How much will the change cost? Also, will this change save money in the overall project?
- The criticality of the area. NO CR/DR can be overlooked if the problem will prove to be mission critical. All other areas of evaluation should be rethought if the bug might possibly create critical problems.
- Other CR under current evaluation. Will another change solve this problem or do other more critical changes rely on this software remaining the same?
- Test requirements. How much testing will be needed which will affect the costs and time needed for the change?
- Resources. Do you have the people available to work on this program? Do you have the hardware equipment available to use for this change?
- CPU and memory impact.
- Benefit. How much of an advantage will it be to change the software?
- Politics. In the corporate and commercial world, it would be good idea to evaluate who is asking for the change and whether or not the board decision might be used as a bargaining point in the future.
- Maturity of the change. How long has the change been before the board? If it is still considered worthwhile to change something after a tong time has passed, then the board should consider it more carefully.

By using these data, you can often minimize the number of “side effects” that the changes will have. Even if the side effects are unavoidable, the use of this carefully documented evaluation process may help to identify where those effects are going to be. Of course, it is now impossible to be sure that all the side effects have been discovered. For example, suppose there are two changes that are being made at the last minute in an emergency situation, and they are each tested and evaluated. Although they may have no real side effects on the system separately, when they are “patched” in at the last minute, they may have serious side effects together. This is the greatest fear when dealing with late patches, but careful documentation and evaluation of the data involved in each change may help to alleviate some of the guesswork.

In the purely commercial arena, credibility is the key in dealing with the marketing division or the customer. If they have a change request that is going to create more difficulty than it is worth, the configuration management people should be able to show *documentd data* that will make the control board evaluation credible. When customers can see the kind of impact a change is going to have on the time, size, or cost involved in a project, they will better understand and more readily accept management's decisions regarding the project. The key is a thorough and well-documented evaluation based on the previous listed data. The list can be changed or expanded on, according to the needs of the project, but as it stands, it gives a fairly accurate picture of the kind of information that is going to be needed for credible evaluation.

8. SCM and the Real World

8.1. Going from the classroom to the corporation

Two points came up early in the meeting that helped to categorize many of the problems discussed later in the day.

1. We live in an irrational world, but computer science and software engineering are based on concrete and rational logic. How does one make this rational knowledge fit into an irrational world? Software Engineering and Design is not just like it is in the textbooks.
2. Very often the existing system dictates what kind of changes take place and what kind of configuration management is used, rather than the ideal or proper software design practices.

Both of these concepts are difficult to teach to young, inexperienced software engineers who are coming directly from the classroom. They are concepts usually learned through experience A software engineering graduate expects to put the principles of Software Configuration Management directly into effect, but is suddenly confronted with an irrational world that does not easily follow the logical course of configuration management tl isn't the main-line textbook problems that are going to throw an educated software engineer; instead, it's the small peripheral problems that build up and take control of a project. These little things, the result of this irrational world, include corporate politics, unforeseeable accidents, human personalities, and day to day unexpected emergencies Also, a new. program manager may have to deal with a system that does not follow regulation SCM practices and does not want to change. Often corporations have become comfortable with a particular structure that does not have room for SCM, and it can be quite frustrating to a young manager to be asked to comply with "company policy' rather than smart software configuration management.

There are some attributes of the irrational world and som e system protocol specifications that will never be able to be changed, regardless of a software engineers chagrin when dealing with them. Learning to deal with these inexplicable and usually frustrating areas of SCM requires experience in the world where they exist. A textbook will never be able to adequately transfer the kind of knowledge needed to deal with the irrational world. There will always be people who will be able to manage corporate software configuration better than others, regardless of classroom performance---another of that irrational world.

A few well educated configuration management personnel are not going to make much of an impact on Software Configuratio Management today. There must be a way to communicate the concepts of SCM to a great number of people involved in the development of software products, even the people who are not directly responsible for the configuration management of a particular system (this would include everyone from the computer scientist writing the actual code to the final consumer of the product). If a few concepts of SCM are known by a majority of people dealing with the development of a software product, then people will be able to function more smoothly within the system and the whole process will be tighter. This is also important when you remember the large number of companies that are using contracts and subcontracts with other companies. Unless the concepts of SCM are widespread among many companies, Software Configuration Management will be dependent upon whether a subcontractor chooses to use SCM or not.

8.2. Two Perspectives

One last point that was emphasized at the meeting was that of two perspectives emerging. It has been previously stated that when a company goes under contract with another to develop a software system, the management people should have respect for the existing SCM structure in the contracting company. The two perspectives are (1) that of the originator of the project and (2) that of the contractor that goes into this program. NASA is a good example. They will often put several companies under contract for a single mission and these companies often turn around and subontract another company to work on variius parts of the system. NASA has a very structured system for configuration management, and the companies under contract often have SCM systems of their own. It is very easy to see the “give and take” needed in a situation like this. Each company needs to try and comply with the SCM demands of the contract originator. A software engineer trying to take the concepts of SCM into the real world should be prepared to deal with these perspectives.

9. Conclusions

It is apparent to me that Software Configuration Management courses are essential to progress within Software Engineering today. SCM is tied to every stage of software product development. A good configuration management team could make the difference between products coming in on time and within cost, and those coming in late, full of bugs, and with greater costs. Education seems to be the place to start, but there seems to be much more involved than classroom development alone. What the group tried to do was begin a program that would teach software engineers that they need to learn the concepts of Software Configuration Management wherever that education may be available (whether learning in the classroom or gaining experience in the field). I would conclude that what seems to be wrong in Software Configuration Management today is that too many software engineers don’t seem to think they are missing much without a solid knowledge of SCM. If they can be shown the importance of SCM, then perhaps they will be more eager to learn its concepts and to use it more often and more effectively in the software development field today.