

Configuration Management

Ian Shere (C) 1997

CONTENTS

Introduction	1
Communications	2
Complexity	2
Newness of Software Engineering	2
The Nature of Software	3
Need for Management and Control	4
Typical problems	4
What is Configuration Management?	5
Identify change	5
Control change	6
Ensure proper implementation of change	6
Report change	6
CM Items	6
CM Plan	7
The CM Process	8
Identification control	8
Project Library	9
Change control	10
Configuration Audit	12
Reporting and Status Accounting	13
CM Tools	13
SCM Standards	14
Design of a Configuration Management System	14
Objectives	14
Major advantages	14
Major disadvantage	15
Identifying Components Types	15
Reporting	15
Selecting A Configuration Management Tool	16
The Future of CM	16
Summary	17
Bibliography	19
List of Figures	19

Configuration Management

Introduction

The development of a software product requires the same process as the development of any other manufactured product, whether it is a motor car, skyscraper, bridge or tunnel. It is from these engineering disciplines that many of the techniques used in the development of software has been borrowed, hence the term 'Software Engineering'. However, the history of software engineering has not been a total success as many projects have either overrun their cost and timescales or had other quality problems of lack of reliability, difficult to use, not suitable for the job intended or just difficult to modify. An example of the extent of system quality problems in the software development industry is given for United States Government software projects in the 1980's. Projects for the U.S. Army with a budget of \$6.2 million, found that only 2% of the software was used as it was delivered, 3% required changes before it could be used, 19% was either abandoned or rewritten, 29% was paid for but never delivered, and 47% was delivered but never used.¹ Although these figures are history, they demonstrate how the software industry needed to improve its development methods.

There are many other examples of software blunders which were potentially life or even world threatening. A software problem caused the launch of a nuclear bomber after a Russian attack which turned out to be a figment of the software's imagination. Another example is the problems with a Space Shuttle that didn't work properly because the computer had been programmed in nautical miles instead of feet. Closer to home examples include the London Fire Brigade, which installed a system which was found to be so inefficient that they thought of going back to a manual system.

Software systems have an impact on all our lives, from the supplying of electricity or gas to our homes to the microwave or video recorder. Clearly there is a need to get it right first time. The cost of developing software systems is high as it is a labour intensive activity. So why is the development of software difficult? The four basic factors that make this so are:

- a) communications;
- b) complexity;
- c) the relative newness of software engineering;
- d) the nature of software itself.

¹ Figures from the NCC Software Engineering Course Material

Communications

A system is developed from an idea or need to solve a problem. This need has to be explained and understood by a variety of people from the customer, analyst, designer to the programmer and made acceptable to the user. English can be vague and easily misunderstood. Misunderstandings between the customer and designer can be brought about by the customer not really knowing what they want or being able to express exactly what their idea is. Once the problem passes to the analyst it becomes steeped in jargon, and when relaying this back to the customer can cause even more misunderstanding.

Complexity

Another factor in making software development difficult is complexity, this has two aspects:

- size;
- inherent complexity.

As hardware becomes cheaper, more capacity for software implementation becomes available, so larger and larger software systems are attempted. Complexity increases simply with an increase in size. This can be seen by considering the number of possible communication paths between 2, 5 and 12 people. The same can apply to modules of code.

The only way to control complexity is by modularisation, dividing the system into subsystems or dividing people into teams, so that communication is restricted to a smaller group.

Newness of Software Engineering

As has been stated software engineering is a relatively new activity. By contrast the areas from which many of the methods for software engineering are borrowed have been in existence for many, many years:

- Mechanical Engineering: more than 300 years;
- Electrical Engineering: more than 100 years;
- Software Engineering: 40 years with major expansion of the activity

in the last 25 years.

The Nature of Software

Software is abstract, it cannot be touched, seen or smelt. It is not a material, it is more ethereal than concrete. This makes it very difficult to control, but very easy to manipulate. Software is flexible, it can be ‘moulded’ to achieve many different objectives or tasks. It is also difficult to understand in the same way as we can understand electrical or physical objects.

Software engineering follows a life cycle (figure 1) in much the same way as any other

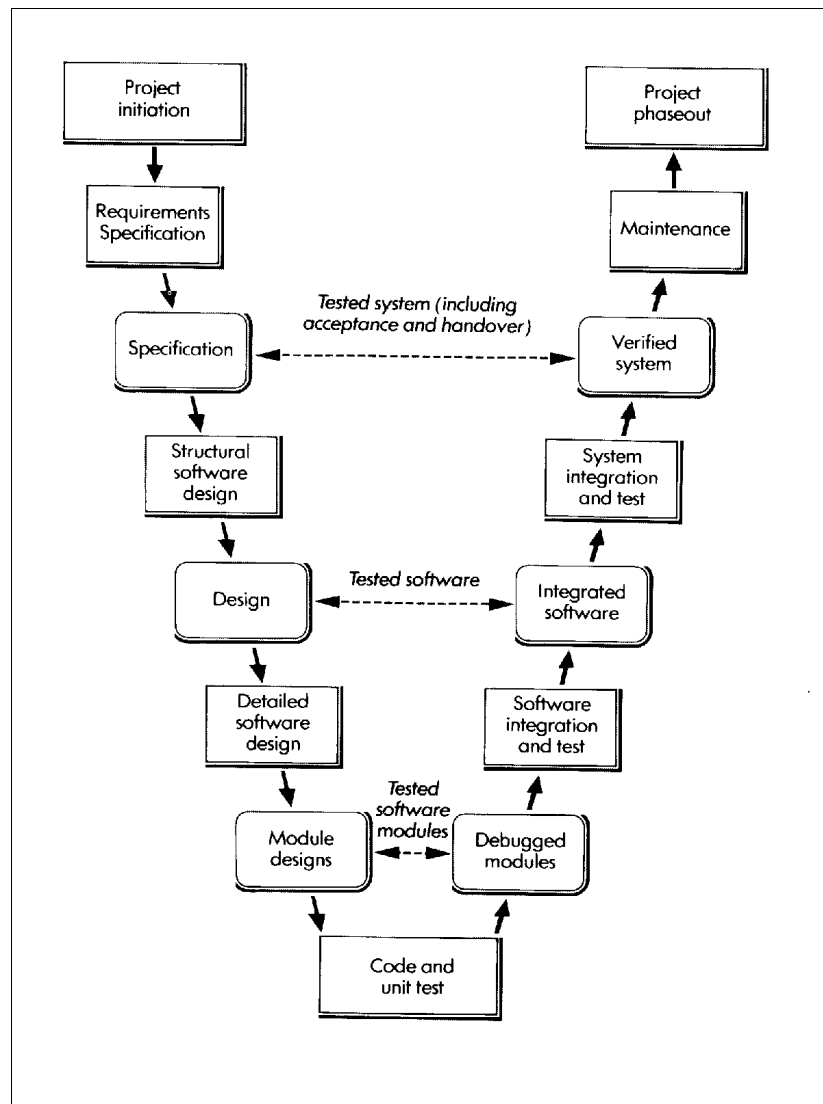


Figure 1: Source: NCC/DTI “SSADM - Achieving Software Quality”, 1990

engineering process.

The aim of software engineering is to develop and deliver systems that have the following characteristics:

- * usable (easy to use, available)
- * reliable (doesn't break down too often, fault tolerance)
- * efficient (makes best use of hardware)
- * correct (accurate, consistent, complete)
- * secure (safe, integrity)
- * maintainable (modifiable, consistent, modular)
- * delivered on time
- * within cost.

This allows the creation of 'Quality' software. Therefore, quality is something that must be built into the product from the start. This is best achieved by ensuring that standards have been identified for all stages of the projects and the products of those stages. The SLC shown in Figure 1 demonstrates the various stages and how cross reference can be made to ensure quality. This very much depends on documentation of the processes and standards required.

Need for Management and Control

A software or firmware product may easily contain many hundreds of code segments. Behind the code segments lie many other pieces of information - specifications, designs, test harnesses, test data, user manuals, etc. All of these pieces, together with the code segments, are the components that make up the product.

As the requirements of the product change, and mistakes are identified in the production process, the components will have to be changed. Unfortunately the causes of change happen in a random or haphazard way, making control very difficult.

Typical problems that arise are:

- * there have been ten changes to a group of code segments when a design change is required. Only nine of the changes were incorporated - one was lost;
- * a customer reports a fault. It cannot be reproduced because changes since the product was supplied mask the fault;
- * a revised product has been sent tot a customer, and the previous copy removed. The revised product goes wrong in a previously correct area. There is no simple way to rebuild the original product, i.e. to undo the change.

Overcoming these, and many similar problems, requires that each component, and each

revision of the component, is uniquely identifiable. It also requires the discipline of maintaining records of changes, and records of exactly which components were supplied to each customer.

A system that provides the necessary degree of control is known as a Configuration Management System (CMS).

The volume of records is such that it is impossible to run a CMS manually. It is therefore necessary to automate parts of the CMS using a Configuration Management Tool.

The Configuration Management System, as well as maintaining a reliable base of knowledge about the components of a product, is capable of assisting the management in many ways, such as:

- * assurance that the procedures are being carried out in the right sequence;
- * tracking the progress of changes;
- * safeguarding components against unauthorised access or change.

Standards such as BS 5750, ISO 9000:2000 and AQAP 13, require that a Configuration Management System is in place and in use. If an organisation approved to one of these standards places a subcontract, then the subcontractor must also meet the standard.

Few organisations will have expertise in the design of Configuration Management Systems, or in the selection and implementation of Configuration Management Tools. In view of the cost, and other implications, it is highly desirable that specialist independent advice is sought.

What is Configuration Management?

Configuration Management (CM) is closely allied with the Quality Assurance (QA) process and function in the development of software. It is an umbrella activity that continues throughout the SLC and has an effect on all areas of the project. It is inevitable that a system will change, not only through maintenance after it has been completed, but also during its development cycle. These changes may occur in the requirements specification, during the design phase, when coding starts, during testing or at any intermediate stage between. The system needs to be managed and change kept track of and recorded. It is for this purpose that a Configuration Management System (CMS) is required so that it can:

- 1) identify change - an identification system to reflect the structure of the product, identifies components and their type, making them unique and accessible in some form;
- 2) control change - controlling the release of a product and changes to it throughout the system life cycle by having controls in place that ensure consistent software via the creation of a baseline product;
- 3) ensure proper implementation of change - by the efficient running of a Project library that collects, monitors, controls and distributes all documentation;
- 4) report change - any change in the change files needs to be reported to all members of the project team.

A CMS affects the process model and users of the CMS in the sense of enforcing policies, procedures and standards on the way users do their work and it keeps track (an audit trail) of how the work is done.

CMS's are still in the early stage of development themselves, although similar techniques have been used in other disciplines. Generally, many of the small to medium projects rely on in-house CMS using some third part tools, even so, CM is considered to the most critical activity in software engineering.

As a definition the following statement made by Babich is offered:

“The art of coordinating software development to minimize ... confusion is called *configuration management*. Configuration management is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes.”²

CM Items

Throughout a software engineering project, many tasks and activities are carried out, each often having a deliverable item. All these items should be subject to CM and recorded in the CMS as Software Configuration Items (SCI). The process starts with the Requirements

² Pressman. R.S.

Specification and includes:

System Description	Test Data & Results
Software Project Plan	Fault Log
Design Specification	Operation & Installation Manuals
System Design	Executable Programs
Program Design	Data Dictionary
Preliminary User Manual	Change Document
Source Code	Maintenance Document
Test Plan	Standards and Procedures
Test Harness	

In some cases it is considered that the software development environment and tools should also be configurable items. This includes versions of editors, compilers, and other CASE tools to ensure that the same versions of development tools are always used on that product and errors that occur due using a different environment can be easily traced.

CM Plan

The CMS plan indicates the scope of the project and provides the main identifier for the remainder of the material. It may refer to the Project Plan, if one exists (at this point) and may also refer to other relevant project documentation. The CMS plan will also have three other sections for detailing management, CM activities and associated material.

The management section will identify, personnel, milestones, baselines, and resources. It should clearly set out responsibilities, for example Board membership. Detail would also be provided on the minimum configuration item (SCI) size, subsystem interconnections and review procedures for changing the status of SCI's.

CM activities detailed in the CMS Plan will identify the procedures for the main areas of CM as given in the CM Process (see next section). These procedures will include rules for naming a SCI and baseline and explanations of what constitute a version or variant together with naming conventions.

The last section of the CMS Plan, all tools, techniques and methodologies to be used in the implementing the plan for the project is stated. This includes CM tools, but is primarily concerned with those tools etc. which will be used for the project itself, and the control of it.

The CM Process

Control over the SCI's is achieved by establishing a number of procedures and processes, which adds to the QA of the software and these includes:

- a) Identification control
- b) Project Library
- c) Change control
- d) Configuration Audit
- e) Reporting and Status Accounting

Identification control: Identification control establishes methods for software documentation, controlling reference and version numbers and making them unique and accessible. It should also ensure that the identification methods are agreed and circulated to all those involved in the project.

Identification control includes the process of setting baselines for the various outcomes of activities. A baseline is

a collection of items which when complete indicates that a milestone in the development process has been reached. A milestone is achieved at some point in time, and the configuration which shows that achievement is the baseline. The milestone is the end of a stage or phase of the project at which one or more deliverables (SCI's) are actually delivered. Any SCI or task can have a baseline, however, the common software baselines are shown in Figure 2. Different authors and authorities on CMS identify different baseline stages.

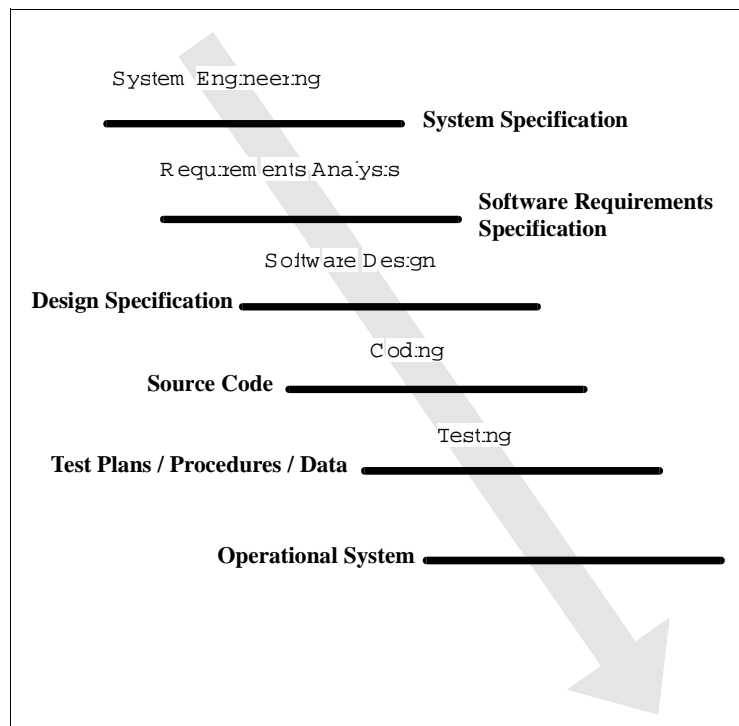


Figure 2 - Source: Pressman: Baselines

However, this can generally be equated to those in Figure 2 or cross referenced as follows:

<u>Stage</u>	<u>Baseline</u>
* System Specification or Requirements Specification	Functional Baseline
* Design Specification	Design Baseline
* Source Code & Testing	Production Baseline
* Implementation	Operational Baseline

During the development it has been noted that there will be modifications and changes. As a result of these requests, the actual actions performed will be determined and the results recorded. One such decision might be to allow the system to reach the Design Specification baseline, keeping track of all requests to that point and then make all changes simultaneously. The important point is that the baselines are all updated as needed so that each does genuinely represent a stage passed. Some modifications, such as the removal of coding or logic errors are unlikely to affect any earlier baselines.

Software and documentation should be bonded when the stage in development has been reached where the Project Management are confident that the software and documentation will only require a minimum number of changes during further development. This point should be clearly identified as bonding will affect what formal procedures will be adopted.

Another consideration of identification, is the establishing of version control. This is covered in more detail under Change Control.

Project Library: Sometimes called a Repository or Project Database. Established before the project commences and creates the procedures for the transferring of SCI's to the library. Other library functions include ensuring software and documentation identity, correct referencing, control of software and documentation distribution, holding standards, procedures and QA records. In essence the project library is the hub of the CMS and is used to record and hold all information on the project including the fault log and change control records. Figure 3 shows the CMS in a diagrammatical form, with the start of the production cycle at the top and proceeding anticlockwise. Initially the library would be empty and in this diagram is represented by the change files in the centre.

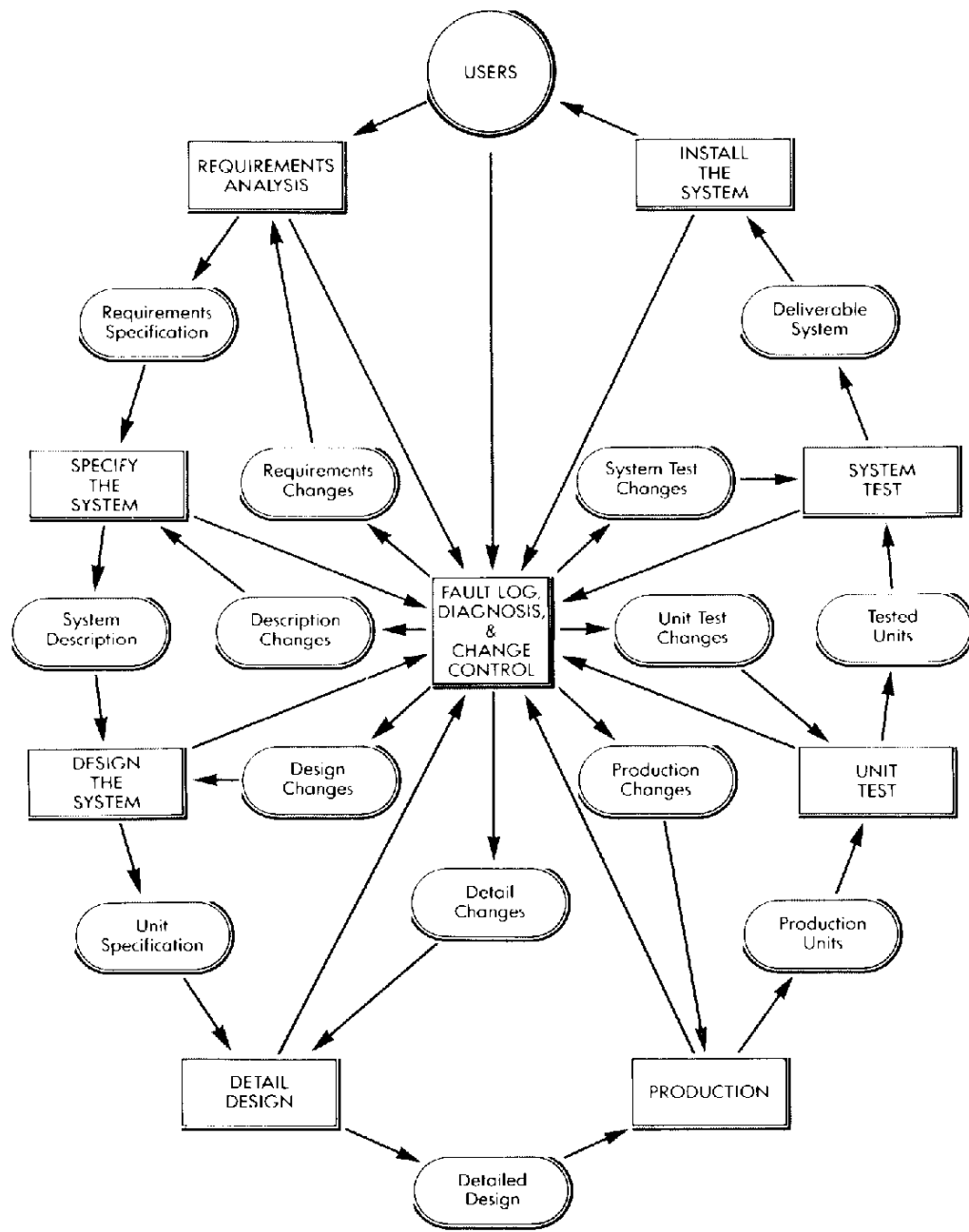


Figure 3 - Source: NCC/DTI "Configuration Management - Achieving Software Quality", 1989

Change control: Change control is probably one of the most important tasks of the CMS. Once a SCI has allocated a baseline, no change should be made without following the complete Change Control Procedure.

For simplicity and initial concepts the change control procedure could be expressed as a flowchart, however, Sommerville, in his book “Software Engineering”, demonstrates this procedure in pseudo code:

```
Request change by completing a change request form
Analyse change request
if change is valid then
    Assess how change might be implemented
    Assess change cost
    Submit request to change control board
    if change is accepted then
        repeat
            make change to software
            submit changed software for quality approval
        until software quality is adequate
        create new system version
    else
        reject change request
else
    reject change request
```

Moving through the above procedure, a *change request* may result for a number of different reasons. The system model will change as it evolves from the outline to the logical description of what is required into a physical description in the form of an executable program, and at the end of any stage during the development and operation a variety of events may occur that will cause unplanned change.

The cause of change could be broadly said to fall into two categories, evolutionary and revolutionary. Evolutionary change is experienced as the system moves through its various stages in the SLC. On the other hand, revolutionary change, ie. unplanned or unexpected changes, can occur at any time in the SLC and during the operational life of the product. Such changes can be caused by the system being unable to satisfy the users' requirements or expectations. This can be generated from new ideas, defect reports or even new laws and legislation.

When a change request has been accepted into the CMS, a *change report* will be

generated by reviewing the change requested. This review will include technical aspects, this will assess whether the change is possible and also consider the consequence or impact of the change, that is if this change is carried out what effect will it have on other SCI's, modules of code, etc. The review will also consider the cost of the change and the impact of the cost and time on the overall project.

The change report would then be passed to the *change control authority* who would, based on the recommendation of the report approve the change and issue a *change order*. It is on the change order that the SCI would be taken out of the library, changed, tested (going through the complete baseline approval process) and then checked back into the library. At this point a new version would be created.

The term version is often used to mean variant, as the SLC develops and baselines are changed (in accordance with procedure and change control) so new **versions** of the SCI or product are created. Generally a straightforward number system is used to identify versions, eg v1.0, v1.11, v2.3a, etc. The time when a version number is moved from the say 1.1 to 2.0 must be decided. This is often based on the nature of the changes made, minor changes to a significant module may move from x.1 to x.2, where as a major change affects the whole produce will move from v1 to v2 etc. It may be that the software is capable of operating across different platforms, or that different modules are available depending upon the task the software is required to do. This would produce **variant** numbers as each variant is capable of operating in a different way.

Configuration Audit: CM does not exist in isolation. It should not even be an 'add-on', it is an integral part of Project Management. Configuration Audit is the par of the CMS which is to do with verification, but it does not actually carry out the process. CM provides the means by which the integrity of the software or product can be assessed. It does not carry out the assessment, because that is done by other disciplines as required by the Configuration Audit (CA). Form the viewpoint of CA a definition of CM could be that it aims to establish and maintain integrity of the product or software by ensuring that it is subjected to all project management disciplines and by controlling changes to accepted and established products.

In essence the CA is the process of checking that CM has been carried out and again assists in QA. The CMS provides an audit trail that can be followed back through each stage or phase of the project.

Reporting and Status Accounting:

This is the final CM procedure to discuss, each of the others providing a specific facility. Reporting and Status Accounting provides the mechanism for being able to report on how the system has evolved and what its status is at any given time relative to the established baselines.

Status Accounting requires that events that cause change in status in either a development or SCI is identified and recorded. It also requires the recording of events resulting from the processing of a Change Request or Defect Report.

Other reports that may be the responsibility of the CMS may include:

- * records of project meetings and action lists (minutes);
- * baseline status reports, identifying baselines and any changes to them;
- * 'change request' and 'defect report' status reports, identifying progress and decisions made;
- * recommendation for new version or variant issue.

The CM process is a continual one running throughout the SLC and touching all areas of software development. The benefits from a good CMS are obvious as it offers monitoring, control, and audit in a way that adds considerably to the QA of the product.

CM Tools

The CM process can be greatly assisted by the use of various third party tools and environments. These tools may be part of the operating system environment and be totally integrated at one end of the spectrum to a simple word process and database at the other. Many tools exist that form an Integrated Project Support Environment (IPSE) or the more modern equivalent of I-CASE (Integrate Computer Aided Software Engineering). These tools allow the creation of the documentation required in a system, eg Object Diagrams, Modular Trees, Entity Life Histories and State Transition Diagrams. From this, an automatic data dictionary can be built. Source code trackers allow the changes made within source code to be documented and recorded. Should faults be identified within a system, then the modern day tools are capable of reverse engineering the whole thing. Notwithstanding, the SE tools on available cannot replace the experience (although this may be rather small compared with other engineering disciplines) and human intuition of the CM manager.

SCM Standards

CM is all about standards, standards lead to quality, software engineering is about producing quality software. The standards used in the CMS can be either in-house standards as laid down in the 'house procedure manual' or it can be taken from the various bodies and organisations responsible for setting national and international standards, eg ISO and BSI. Current standards applicable to CM are DEF STAN 05-57/2, IEEE Standards 729-1983, 828-1983, 1042-1987 and various US Military and Defence standards.

Design of a Configuration Management System

A Configuration Management System (CMS) is just another system, albeit one for use by system developers and maintainers. As such, it should be designed, constructed, installed, and maintained in accordance with the installation's procedures and standards.

Since it involves an area where not many installations have experience, the following notes are offered for guidance. They are intended to supplement the procedures, not supplant them. However, it may be advantageous for specialist help to be sought, e.g. through consultancy.

Objectives: The primary objective of a Configuration Management System is to assist managers of production and maintenance projects in enforcing the disciplines and agreed procedures to control the changes in the systems and hence maintain the integrity and traceability of the system.

To this must be added local objectives covering such items as cost, time, machine resources, etc.

Note carefully that the CMS can only assist the management. Poor management practices, or lack of quality control, will prevent the best designed CMS from fulfilling its objectives.

The CMS overlays the production and maintenance life cycle of the systems. Organisations using more than one life cycle model should consider carefully whether they require one or more CMSs. The major advantages of only one are:

- * consistent operating procedures throughout the organisation;
- * easier consolidation of reports.

The major disadvantage is:

- * an excessively long list of component types which may lead to confusion.

Identifying Components Types: Whether the CMS is to cover one or more life cycle models, the next task is to identify all the component types involved. This list can be quite lengthy, but must be comprehensive.

For each component type it is necessary to identify:

- * how many components are there likely to be of this type;
- * how much space will they require;
- * how frequently do they change;
- * are they Configuration Items;
- * is the information sensitive enough to warrant Encoding;
- * who needs to see the components;
- * does seeing the components need to be authorised each time;
- * if so, who authorises;
- * who needs access to change the components;
- * does access to change need authorisation;
- * if so, who cancels;
- * who needs to insert or replace components;
- * does this need authorisation;
- * if so, who authorises.

Reporting: Considering the CMS as a whole, the other area to be considered is that of reporting. Reports are going to be required by:

- * Project Leader/Manager;
- * Senior Management;
- * Quality Assurance;
- * Audit.

The reporting requirements of the various groups will enable decisions to be made about the data that must be collected at each Authorise/Extract/Insert/Replace event.

By this stage it should be obvious that the volume of data is going to be quite considerable. It is most unlikely that an entirely manual system could succeed, and so part of the process must be automated, using a Configuration Management Tool.

Selecting A Configuration Management Tool: The need for Configuration Management Systems is so widespread that a number of companies supply generic Configuration Management Tools. These tools need to be tailored to suit the particular installation by, for instance:

- * identifying the component types;
- * specifying the relationships between the component types;
- * identifying tools or work benches used to transform between pairs of component types;
- * establishing the security requirements.

Tools may be obtained as:

- * part of the operating system (MAKE & SCCS are part of UNIX);
- * stand-alone tools;
- * part of an Integrated Project Support Environment (IPSE).

Information about selection of particular tools will be carried out in the same way as selecting any Software package.

The Future of CM

CMS have evolved around in-house systems and have been supplemented with manual procedures and policies. To days systems have a better understanding of the need for CM and the tools available to them. However, the current situation of ad-hoc CM supported by various third party tools each for a different area of the overall CMS must surely indicate the possibility of bringing all these areas into one system.

There are many political and technical issues that affect the future of CMS's. An example may be of the political issue concerning the development of CASE tools. For instance, should CASE tool suppliers ignore CM within their tools and assume that operating system or environment vendors will provide the framework and support for CM? Or should CASE tool builders provide support withing their tools for CM? If vendors include their own CM support, users will have to solve the problem of integrating different CMS's when they install their (or different) CASE tools. Also, from the vendors' viewpoint, will they essentially be duplicating much of the work that has already been done or is not being attempted for environment frameworks? This political issue continues.

From a technical viewpoint, many research issues will affect the capabilities of CMS's. These include:

- What is the appropriate technology on which to base a CMS? Is an object-oriented database with persistency notions for constant objects the most suitable?
- In what layer of an environment's architecture does CM fit? Should it be at the base level in the database, making it an integral part of an environment, or is it all a matter of specifying CM as a process at a higher level in the architecture?
- Can the mechanisms for CM be separated from the functionality of CM, in other words is there a common CM model?
- Is it possible to support cross-development of software? Can engineers develop a product on a host machine easily move it to the target machine while still maintaining CM control over the product?
- Is the scale or size of a project a factor for CMS's? Is the CMS for a million line software product the same as that for say a 100 million line software product?
- Is it possible to fully model all aspects of the CM process including the people intensive areas into a CMS?

These questions and many others need to be answered before the future of CMS's is decided. As usual the answers to these questions probably lie with those who wish to profit from the development of such CM systems rather than those who wish to build 'quality' software systems.

Summary

Software engineering is a relative new art (rather than science) with a somewhat chequered, albeit, short past. As a discipline it relies on concepts and practices from other engineering environments and is continually frustrated by the complexity of the systems it has to build and the confusion and misunderstanding caused in its communication with the customer or user.

In Software Engineering there is the opportunity to avoid making the same mistakes again,

and in many software engineering projects, it is more important that the mistakes are avoided than it is in a routine 'data processing' development. The cost can be enormous, and there can be both lives and livelihoods at stake. CM is the management of the evolution of a software product. It is one area in software engineering where clear and defined management principles and procedures are laid down. Its purpose is to control the process of change and in doing so provide a historical record of the progress of a project.

CM itself has evolved and is now surrounded by a plethora of tools to aid the CM manager. The future of CM is as uncertain as the future of any other process in the development of software. As hardware becomes more powerful, software more readily available and the visions of the software engineer, some of the political and technical issues surrounding the further evolution of CMS's may be resolved.

Bibliography

Roger S Pressman, Software Engineering - A Practitioner's Approach, European Edition, McGraw-Hill, 1994

Ian Sommerville, Software Engineering, 3rd Edition, Addison Wesley, 1989

Richard Barker, CASE*METHOD - Tasks and Deliverables, Addison Wesley, 1990

Grady Booch, Software Engineering with ADA, 2nd Edition, Benjamin/Cummings Pub., 1986

Software Engineering, Student Notes, National Computer Centre, 1987

Various NCC/DTI 'Achieving Software Quality' booklets, 1989 - 1990

Susan Dart, Technical Report - "Spectrum of Functionality in Configuration Management Systems", Software Engineering Institute, Carnegie Mellon University, USA, 1990

Susan Dart, Concepts in Configuration Management Systems, Software Engineering Institute, Carnegie Mellon University, USA, 1992

List of Figures

1: *Source: NCC/DTI "SSADM - Achieving Software Quality", 1990*

2: *Source: Pressman: Baselines*

3: *Source: NCC/DTI "Configuration Management - Achieving Software Quality", 1989*