Software Configuration Management Tools

Angus K.F. Chan
City University of Hong Kong
Department of Computer Science
83 Tat Chee Avenue, Kowloon, Hong Kong
kfchan@cs.cityu.edu.hk

Sheung-lun Hung
City University of Hong Kong
Department of Computer Science
83 Tat Chee Avenue, Kowloon, Hong Kong
cshung@cityu.edu.hk

Abstract

Software Configuration management is concerned with the identification, organizing, and controlling the configuration of and changes to a system under parallel development environment. It should encompass all system components. However, most existing software configuration management tools have emphasized too much on version control of source files and have neglected some of the other vital functionality. In this paper, recommendations are made to the desirable features that should be incorporated in future tools to make them truly effective in the software development process.

1. Introduction

Software Configuration Management (SCM) is concerned with the identification, organization and control of the software components known as configurable items in a software system under a parallel development environment. The configurable items should include all system components such as source codes, documents, test plans etc. The goal of SCM is to maximize software productivity and improve quality by minimizing mistakes [1]. Software quality is ensured through managing changes that occur during the software development process. IEEE provides definitions for SCM activities known as Configuration Identification, Configuration Control, Status Accounting, Audits and Reviews [2]. As a result of the lack of a general framework to guide software engineers in their implementation of tools to support the defined SCM activities, there exists a great deal of confusion in the interpretation of the terminology and functions. The problem is further complicated by most of the tool designers whose concern is with larger scale software development. It is the belief of the authors that SCM has an important role to play for small and medium size projects and the definitions of the SCM activities should be modified and be extended to provide more concrete but at the same time more flexible guides to all SCM tool designers. This paper aims to review some of the important SCM activities and discuss the relevant implementation issues. In the following, the authors attempt to identify some of the problem areas associated with standard SCM activities.

1.1. Identification

One of the basic requirements of SCM tools [3] is to provide information on all software configurable items. The information includes description of each configurable item, linkages and interdependencies with other items. Identifying the correct item is important for many reasons. For example, if the wrong version of a software item is being used, bug fixes in a program can quite easily resurfaced when the older version of the item is inadvertently used with other modules. Items interdependence are especially important during system build where the software items are combined and compiled into derived objects. The interdependency information can be used as blueprints for the construction of derived objects, e.g. the description may be the version number and the options of the components to be used in the particular system build. In the case of a distributed system build, the problem is complicated by the need to combine objects which may reside into different physical systems into an executable object. Identification has been implemented in a number of SCM tools but their effectiveness have rarely been evaluated. Some of the observed problem areas with these tools include:
1. Most of the current tools focus on identifying the correct version of source and object files. However, few facilitate the identification of documents such as specification and test plans.

2. It is important to understand the structures of software objects. If a constituent item of a software object is changed, all affected items should be changed. A kind of bill of material is required to facilitate developers' awareness of the inter-dependencies between the configurable items. Graphical support to display the structural dependencies should be most valuable but is currently unavailable in the market.

1.2. Configuration Control

No system developments are immune to changes. Changes to software configurable items should be properly controlled and monitored. Changes can occur not only to the source and executable files, but to other configurable items such as documents. Change requests from members of the development team should be vetted by a change control board (CCB) which either approves or disapproves the changes. Current SCM tools have very rigid approval procedures and change control board composition. An area for improvement should be to allow flexibility in the composition of the CCB and the procedures to reflect the size and scope of the software project. For medium to small projects, the substantial delays and overheads incurred are normally unacceptable. To further expedite the change request process, an efficient means of communication is required. CCB and development team members may reside at geographically dispersed areas and a good email service is required in support of the change request process.

Another essential element of configuration control is the check-in/out facility. The check-in/out facility controls the manner configurable items are being modified. Software development normally involved a team working concurrently. Without proper management and control of these concurrent activities, to maintain the software system's integrity [4] is a difficult if not impossible task. An Example of the problem associated with parallel software development is in the modification of a source program by several programmers with one programmer accidentally destroying the work of another programmer. Parallel software development can involve a development team spatially dispersed over a wide area but sharing same source files and derived objects residing in computers with different hardware and software configurations. SCM tools should provide remote access over heterogeneous computer networks [5] and to provide distributed transaction management for the shared software objects [6]. Logical changes to programs such as bug fixes or enhancements are usually complex and slow, traditional transaction management models are probably too costly and some new forms of conflict management are required. An example is the more elaborate transaction model. Up to date, there is no commercial implementation of well designed distributed transaction models in SCM tools.

Many authors [7], [8], [9] has made many discussion on version and revision control as it is the heart of configuration control. Unfortunately, there is still a great deal of confusion on the usage of the terms revision control and version control in most commercial tools. Revision should refer to the sequential evolution of a module. For example, when a module is edited, the result is known as a revision of that module. On the other hand, a version represents a variant of a given module for another application. A release occurs only whenever a version of the module is ready for sale as a product or has become official. Of the many features in configuration control, traditional SCM tools have focused mainly on revision, version and release control but isolated from other aspects of software development [10]. One of the major weaknesses of all existing tool is the integration of version and release management with the development tools and environment. A seamless integration can streamline the development process and improve productivity as developers are not forced to waste time on the SCM tool.

1.3. Status Accounting

SCM tools should keep a continuous record of all changes and problems to insure against disasters as well as to ensure the quality of the software development process. However, there is confusion as to the type of information to be recorded in such a process. Most traditional SCM tools which claim to have implemented the function offer nothing more than simple information such as the release, level, date and time of creation, etc. In the authors' view, they are inadequate and is of little use in the quality assurance process. Few SCM tools actually keep track of changes to configurable items in terms of defects and enhancements for productivity and project progress assessment. The process of logging and displaying the relevant information should be highly efficient so as not to create overheads on the development process.
1.4. Configuration Audit

Proper authorization procedures should be in place to ensure the accuracy of the product. Configuration audit helps to ensure the software product is as specified by the system requirement. The software development process should be verified, validated through formal reviews and inspections performed on all source files and documents. Mechanisms to automate or semi-automate the review and inspection process should be an interesting research area which are not currently supported in most commercial tools.

1.5. Vendor Control

Subcontracting, outsourcing, the use of shareware and reusable components are fast becoming the major part rather than a trend for the software industry. Few SCM tools have built in facility to control the usage of third party software. Vendor Control refers to the monitoring and control of all third party software and extends to all relevant documents, plans, tools and tests etc. There is currently few guidelines or discussions in the literature on how this should be carried out. With their over growing importance, there is great urgency in formulating some formal methodologies in the area.

Of the above mentioned SCM activities, all earlier SCM tools tend to emphasize on version/release (or revision) control of source files. Well known examples are MAKE (A Program for Maintaining Computer Programs) [11] and SCCS (The Source Code Control System) [12]. To a certain extent, they have also control over some of the documents, but they do not possess full documentation control functions. The identification function in these tools concern primarily with identifying the revision numbers of the source files. As a rule, the identification of the inter-relationships between software configurable items (including library functions) is non existing in any of the tools. The Revision Control System RCS [13] differs from others in its change control feature in the form of check-in/out facility. RCS also distinguishes itself in its ability to freeze a revision for baselining [14]. Current SCM tools have increasingly focused on the importance of distributed software developments across homogeneous as well as heterogeneous computer networks [15]. However, distributed transaction management other than simple locking is absent from nearly all commercial implementations. Although a number of tools have facilities for status accounting, they are rarely comprehensive enough to offer much help in improving the quality of software products. With few exceptions, most SCM tools offer no support in the areas of configuration audit and vendor control.

In section 2 of this paper, a survey is presented of some of the popular SCM tools and their main design features. Direct comparison between SCM tools is difficult as each tool tends to focus on SCM activities and adopting different terminology. In section 3, a functional evaluation based on a measure of the expectation from the users' viewpoint is presented with recommendations on some new design features. In section 4, the conclusion is drawn with suggestions on future work.

2. Survey on Current SCM tools

Since the establishment of Software Configuration Management as a discipline of Software Engineering, many SCM tools are available commercially. However, the majority have over emphasized on version and release control of source files but have neglected other equally important requirements of SCM. An example is configuration audit which has received little attention to-date. In this section, the authors present a brief introduction to some of the more popular SCM tools currently available in the market. The SCM tools can be broadly classified into first and second generation tools. The first generation tools focus mainly on version and release control whereas the second generation tools pay much more attention to network support for parallel software development.

2.1. First Generation SCM tools

Of the many first generation SCM tools, three tools namely SCCS, MAKE and RCS stand out as the most important. Other than simple version/release controls, they all have unique features which cannot be found in similar tools of the same generation.

2.1.1. The Source Code Control System (SCCS) [12] : SCCS is a UNIX based source code revision control system. It can store multiple revisions of source files in the form of an ancestral tree structure. It is one of the early systems which incorporated change control and status accounting features.

Its Change Control Mechanism controls changes to source files through a couple of security features. The SCCS's protection facilities control which modules a programmer can add to, and in which software release. SCCS keeps a list of releases that should be locked for each module. Consequently, programmers cannot add to the locked releases. Some implementation may in
addition keeps a name access list for each module or maintains release passwords to control the addition to a password protected release. SCCS requires an administrator's password in order to perform administrator functions such as setting locks and passwords, changing the name access list, creating or deleting modules etc. SCCS treats each software module as a set of related sequences of source code, each member represents one version of the module. Versions are consecutively added into the set as each time the original module was changed (the term versions should be treated as revisions). AS all versions of a module are stored together in the same file, storage space requirement is extremely high. In order to save storage, source codes common to more than one version is not duplicated. SCCS makes use of the concept of delta storage [16] to achieve this end.

In order to determine the correct version of the source code, SCCS places identification information as a set pre-defined identification keywords within the source code provides. Each keyword represents one piece of information, e.g. R stands for the release, L stands for the level, D stands for the date, etc.

Status Accounting is another distinct feature of SCCS is documentation generation. It automatically records the time when the programmer adds a new delta, the places of source lines change. However, the reason for the delta change must be provided the programmer manually. SCCS makes use of this documentation to provide several reports, e.g. whenever a module is accessed, SCCS display a chart that shows the release, level, date and time of creation, who added it, why added etc. This information is useful when something goes wrong with a program. Status accounting is achieved through logging this information for each revision.

SCCS is often combined with the SCM tool known as MAKE to expand its functionality.

2.1.2 MAKE [11]:

MAKE allows the programmer to maintain, update, and regenerate groups of computer programs. It is a simple tool by itself and are normally used with other SCM tools. It’s adaptability explains why it has been widely used by UNIX programmers since 1975. One of its most distinct features is the provision of software structure identification (information on the interdependence between software items) in support of its version control mechanism. A software system is normally composed of a group of dependent computer programs, e.g. program file B may depend on file A. If a change occurs in B, A may be required to be updated. MAKE maintains the dependencies through a file called makefile. It contains a description of the dependencies in form of a dependency line between a target and a source. Each dependency line is followed by one or more commands. Whenever a source is newer than its corresponding target, MAKE executes commands in the makefile to update the target. By keeping track of the status of the components in the system, it ensures the final system is built from the correct component pieces. The makefiles can also act as the configuration specifications by providing information such as how and what compilers should be used to compile a particular source file. It can synchronize the source with its object with auto-compilation.

It supports an auto-checkout (a very simple form of change request control). It is often called a front-end tool because of its common usage with other SCM tools.

2.1.3 Revision Control System (RCS) [13]

RCS is UNIX based SCM tool that provides revision/version control. Revisions are organized into an ancestral tree that permits easy management. One of the powerful features of RCS is that it provides several methodologies in its implementation of delta storage. One of its distinct features is Check-in/out control and baselining.

It supports check-in/out function...xx but not really suitable for a multiple developers environment as lock overriding (breaking) is permitted. Simple security control is implemented through access lists. Revisions can be frozen for baselining..xx.

Logging is supported in support of simple status accounting. RCS can be combined with MAKE to handle configuration processing.

The following is a list of a few SCM tools which are little more than simple revision control systems:

2.1.4. CLEAR/CASTER [17]

It is one of the earliest of the SCM tools. It provides simple revision control on the source files together with documentation and messages. This tool provides control over the whole developmental process.

2.1.5. Software Development Control System [18]

It is a very simple revision control system with no more than 2 revisions at a time. A complete log of the
old revisions are kept as a form of status accounting. It is sometimes used with MAKE.

2.1.6. DEC Code Management System (CMS) [19]
A very simple revision control system for DEC computers. It manages tree-structured revisions in groups. It is often used with MAKE.

2.1.7. Others
ENDEVOR/MVS works in the MVS/TSO environment with text-mode interface only. It main distinctive features are change request processing, configuration control board (CCB) and very good security control. auto-compilation (synchronization of source and object) is another important feature but it suffers from the lack of user-friendly GUIs. Lifespan supports packaging implying that related components can be automatically packed into a package. Change and Configuration Control/Development and Maintenance CCC/DM is a tool which supports four configurations: development, test, approved and production. It does not allow modification of any previous released items. Another important feature is the support for emergency changes.

2.2. SECOND GENERATION SCM TOOLS

2.2.1. Aollos DSEE [20]
Domain Software Engineering Environment (DSEE) is designed to manage large-scale development where data are normally spread among many locations. To support this distributed network computing environment, it provides configuration management (system building) and history management (source code control) to satisfy the need of programming-in-the-large under the parallel environment. It aims to fully utilize the computing resources distributed over a wide area network. DSEE provides very efficient support for parallel software development in a network environment.

System Builder
System building can be performed in parallel. In DSEE, system building is based on three principal concepts:

A system model that describes the components of a system and their dependency relationship. It also provides sufficient information to decide which components can be built in parallel.

A configuration thread that describes the version of each component to be used for a particular system build and specifies the options to be used during translation of each component. A configuration thread tells us which revision one is working on in a particular revision group (e.g. A may be working on revision 1.1 and B on revision 1.2 of the same revision group).

A derived object pool that contains a collection of arbitrary versions of binary files and other objects produced by translators during system building.

When you want to build all or part of a system, an appropriate system model and configuration thread are selected. The configuration manager combine the system model and configuration thread to produce a desired version description. This description is used to search the derived-object pool. If a component is found, its associated derived objects are reused; otherwise, an appropriate translator is invoked to rebuild the component in accordance with the desired version description.

Building Arbitrary Configurations in Parallel
The source code control system (DSEE history manager) is closely integrated with the system builder to support rebuilding a component according to a specific version description. It uses delta system to save storage space for source element. It enables transparent access to any version of a source element by using a per-process version map, which allows simultaneous building of different system versions from same set source elements. This ability is essential for building arbitrary configurations in parallel.

Parallel System Build
DSEE includes a parallel builder which support concurrently building several components by using the available network computing resources. When the configuration needs to build several components, it starts a parallel building process. The scheduler continues to submit builds in accordance with the partial ordering of the system model as long as there are buildable components and available build servers. The builders on remote computers automatically read source codes from various file servers and write the results to file servers that contain the derived object pools. Overall status and progress of the building process can be seen from a continuously updated graphics display.

Other than the parallel build features, DSEE is very similar to SCCS in functionality. It also provides a mechanism to alert maintainers about the need for system rebuild after a change.
2.2.2. Micro-focus & Intersolv PVCS Option [21]

PVCS is a well-known tool for software configuration management with the LAN environment. It consists of a family of tools that gives you capabilities for version management, reporting, problem and change request tracking, configuration building, promotion and transfer to production. The family consists of Version Manager, Configuration Builder, Reporter and Production Gateway.

**Version Manager**

The Version Manager has a good document handler that supports version management, version management and promotion management, access control (security), item locking to permit parallel development, very comprehensive status accounting facilities and delta storage between revisions. An additional feature is its ability to allow developers to add services (user configurable). For revision management, it helps manage the revisions of any program source, executable, graphic or text or binary file. Any change to the system is automatically organized and recorded. For version management, it clearly defines and manages versions of program builds. For promotion management, it provides the ability to define your project in a promotion model. Consequently, programmers can know the status of their code in the development, and manager can control the movement from one step to the next and to ensure that fixes are tested and included in the production system. The version manager also provides support for code sharing across all development platforms, and locks across the network are possible to prevent unnecessary overwrites.

**Configurable Builder**

The Configuration Builder is a system constructor that automates the process of reliably reconstructing a software system. It eliminates build errors by using a built script that enable repeatable, automated builds of your software development projects. One of the most important feature of the configuration builder is 'footprinting'. It is something like putting remarks in source programming. It is automatically added to source codes. The problem of tracking is made easier by providing this historical data.

**Production Gateway**

Production gateway permits multi-platform operation. It can be used with IBM's Software Configuration & Library Manager (SCLM) and other IBM products.

2.2.3. Parallel Software Development (CVSII) [33]

Besides revision/release control, UNIX based CVSII distinguished itself for its multiple developers support. It allows concurrent access with build-in conflict resolution. It has Check-in/out facility but no user request control. It cannot remember 'who' has checked out a copy of the source. It is one of the few systems that provide subcontractor (third party vendor source) tracking. User configurability is another one of its important feature. One of the major weaknesses of CVS II is the lack of security on the source repository and baselining and is often used together with RCS.

2.2.4. Network Software Environment (NSE) [22]

Multiple developers can transparently access the same configuration and to work in their private workplace (child environment). These private workplaces are isolated from each other but their relationships are maintained through NSE. It offers good team work support since each developer can work in its workplace but NSE can still coordinate and synchronize the change in the workplaces to the parent environment.

2.2.5. Parallel Configuration Management System (PCMS) [23]

PCMS works under UNIX, Ultranex and VMS environments. Besides providing revision/release control and tracking (status accounting), PCMS was designed for heterogeneous distributed networks (LAN) environment with client server support with ORACLE as the relational database management system. Other than parallel development support, it offers additional features such as:

- Documentation change control.
- Security and integrity are enforced.
- Change requests control in place through emails for feedbacks and event processing.
- Status accounting is provided with fault reporting facility.
- Check-in/out facility for all modified items.

PCMS has many nice features but the lack of WAN support is one of its major weaknesses.

Most of the first generation SCM tools focus mainly on version/revision control whereas the second generation tools distinguish themselves from the first generation SCM tools in the parallel development support.

3. SCM Tools Evaluation
3.1. Identification

One purpose of identification is to provide the ability to recognize, maintain and access the correct version of source and object codes. Most tools that provide this function are very similar in nature. The source and object code are normally stamped with predefined identification strings and keywords that contain the essential information of the configuration items. One typical string may contain the revision number, date, time and author. An example is RCS which identifies items through a stampping string containing the revision number, date, time and author. In some cases, the information can actually be customized to the user environment [13]. SCCS identifies the correct version of source code item by placing the identification information within the source code itself which will appear when the module is loaded at runtime. SCCS has a set of predefined keywords for pieces of information such as release, date etc.

In the identification of the correct version of source items and the dependency between them (e.g. RCS), there are no shortage of SCM tools. However, there is a general lack of tools to link the source items to non-source items such as design documents, test cases and test data. This capability is vital to the developer for ensuring making the relevant changes to the documentation whenever there is a change to the source item. In addition, it is also important when an item is tested, the system should allow team members to identify the correct test plan and test cases.

3.2 Configuration Control

3.2.1. Version Management

a) Versioning

The term versioning actually implies parallel development support whereby the system is evolved into variants to fulfill user requirements as a result of different operating environments. To support this mode of development, the SCM tool should provide branching capability to facilitate with ease in the creation of variants from the main development line (the trunk). If necessary, the variants should be allowed to merge back to the trunk. SCCS, RCS, PVCS and DSEE all have simple commands for branching the individual items from the trunk and merge back to the trunk. They also provide commands to compare and print the differences between versions of the same configurable item. NSE views branching in a much more general context by treating it as a function that applies to different versions of the system. In making a branch to a particular version of the item, changes including those of other versions are reconciled and merged back to the original environment [24]. The advantage of such an arrangement is that with other child environment, the change can be made immediately available to others after merging. This achieves effective synchronization and sharing of the changes.

b) CM Model

There are currently four different kinds of version management adopted by SCM tools. They are commonly known as Configuration Management (CM) models. Detailed discussions on these models can be found in Feiler [25]. They are:

- Check-in/Check-out Model
- Composition Model
- Transaction Model
- Change Set Model

Of the four models, only the first three models are implemented in commercial systems. Check-in/Check-out model provide independent versioning of the individual configuration item in the system. Concurrent modification of configuration items are facilitated through the use of locking, branching and merging. This model is the most widely adopted in the earlier commercial systems. The composition model improves on the check-in/check-out model by adopting a two steps approach. In the first step, a system model is built to provide information on the composition and structure of the system. Version selection rules are then applied to the system model to choose the required version of the system. Transaction model manages system evolution as a whole. The result is a series of atomic changes for each version of the system configurations (transaction model needs further elaboration). Transaction model provides better support for concurrent modification in a team environment.

UNIX based tools such as SCCS, RCS and PVCS adopt the check-in/check-out model, DSEE is a typical example of the composition model and the transaction model is best represented by NSE. Check-in/check-out tools provide good version management for individual configuration items. It concentrates on the management of the repository but fails to provide support for coordinating the users' workspace. This places severe limitation for team development.

C) Storage

A common problem associated with version control is the requirements to store all versions of each configurable item. In order to save space, the technique known as delta
storage is employed to better utilize storage. SCCS and 
RCS create delta based on longest common sub-string 
proposed by [16]. However, this method cannot produce 
minimal delta because it does not cater for crossing block 
move. DSEE uses algorithm proposed by [26] to detect 
crossing block move and hence produce a better delta. A 
faster algorithm was proposed by Tichy [27] to produce a 
more minimal delta.

There are two techniques adopted for delta storage: the 
forward delta and the backward delta. In forward delta, 
system stores the complete first version but only the 
differences between successive versions. In backward 
delta, system stores the complete latest version but the 
differences of all previous versions. Backward delta saves 
time for accessing new versions whereas forward delta 
requires less space when there are many lines in a delta 
tree. SCSS stores items in forward delta, whereas RCS, 
PVCS, DSEE and NSE all use backward delta. As access 
time rather than space is of more concern to most tool 
designers, it is obvious that backward delta is the 
preferred choice for the more recent tools.

D) Project Structure

Hierarchy of directories are very useful means of 
managing project files. Large projects are normally 
composed of many sub-projects and it is important that 
developers can concentrate on their sub-project but can 
at the same access system codes of other sub-projects for 
system builds and integration testing purposes. However, 
it is interesting to find that few SCM tools support such 
project structures, e.g. SCCS, RCS and DSEE all do not 
support hierarchical project directories. Others such as 
PVCS, which appear to support hierarchical directories 
are in reality flat directories systems. Versioning should 
be performed on all items of a sub-directory as a 
composite.

3.2.2. Change Management

It is of vital importance that any change to the 
repository should be tightly controlled. Change request 
should be subject to authorization to determine either to 
approve or reject the change. Change request control can 
also provide good source of information for tracking 
changes. SCCS, RCS, DSEE and NSE all do not provide 
change request mechanism. PVCS is one of the few tools 
which provide comprehensive change request control. It 
also provide requesters update information on the status 
of his/her request.

Change request besides providing the means of 
submitting, controlling, evaluating and tracking of 
changes, should have the capability of modifying 
multiple source, header, program documentation and 
test data together. In other words, the SCM tool 
should allow the change requester to group related 
items to be modified as a single logical unit.

3.2.3. Access Control/Promotion Level Control

a) Access Control

All changes to the repository of a software system 
should be guarded against unauthorized or accidental 
access. Users and groups of users should be assigned with 
specific privileges and restrictions at both project and 
individual item level. Access rights should include read, 
modify, add and delete. SCCS, RCS, DSEE, NSE have 
additional security features on top of what is provided by 
UNIX, e.g. RCS has its own access control list for 
individual item and only users on the list can perform 
update operation on the item. PVCS is a tool with all its 
own sophisticated security features. It provides a set of 
prefixed base privileges and restrictions. For groups 
with similar privileges and restrictions, one can predefine 
composite privileges and restrictions in an effort to 
simplify the procedure for defining access rights. One of 
the major security fault of PVCS is that file archives can 
be accessed directly outside PVCS.

Security control is in general weak for PC and 
UNIX based SCM tools. For projects with 
geographically dispersed team members, better 
security is required.

b) Promotion Level

Software development goes through well defined 
phases known as the life cycle. Promotion level defines 
the progression from one phase to the next phase. It is 
important that control should be applied to items at each 
level, e.g. only software testers can access test level items. 
SCCS, RCS, DSEE and NSE all do not provide 
mechanism for defining promotion levels and PVCS is 
the only one that supports such a function. The concept of 
promotion level works well at the project level but is 
impractical when the items are shared between several 
projects. A confusing situation will occur when the same 
item is being used by different projects at different 
promotion levels.

Promotion level control for shared projects has yet 
to be implemented in commercial tools.

3.2.4. Release Management

Release specify the collection of specific revisions of 
configurable items which construct a particular software 
system to be distributed to users of the system. There are 
two aspects of release management to be considered,
namely Release Production Control and System Delivery. Release production control refers to the identification of all configurable items for a particular release and the tracking of all releases. It should ensure that all items of a release has the full approval by an authorized body such as the configuration control board (CCB). In addition, records should be provided of the inter-dependence among different software systems’ releases. System delivery on the other hand is concerned with the distribution of the right releases to the target users. It must ensure that all items in the configuration list are properly packaged in the software release. None of the tools to-date pay much attention to release management. A primitive form of release control is provided by PVCS in the form of version label to identify software releases.

In general, there is weakness in software release delivery. Most of the delivery system to-date is manual in nature. An ideal scenario should allow developers to semi-automate the process of packaging the correct versions of items for a specific release. The items should include not only the software items but all the relevant documentation. The software release should be distributed directly and electronically through Internet. The system should support transparent browsing and retrieval by users irrespective of their physical locations. SCM tools should preferably have remote installation capability to allow the automated installation of new releases on sites of the customers.

3.2.5. Automatic System Build

System building refers to the process of collecting the correct source items and transforming them into the required system. The process should preferably be made automatic to minimize or even eliminate human errors. System builds in most of the tools involve the use of build scripts which define system architecture and the inter-dependencies among the items. Earlier tools such as SCCS and RCS all lack system build capabilities. They are often used with MAKE to support system build. PVCS’s configuration builder is similar to MAKE which makes use of build scripts for automatic system construction. The dependency scanning feature of PVCS allows the computation of item dependencies which can be used to update the build script for system construction later. DSEE went a step further by combining its system build model with the configuration thread to obtain the Desired Version Description for the release. The description is used to search the derived object pool for the desired object items. If the item is found, it is reused whereas the item will be retrieved from the repository. DSEE also provides the outstanding parallel system build feature for concurrent system builds.

Configurable items are normally distributed across a network, but with the exception of DSEE, few tools provide the support needed for distributed system builds.

3.3. Development Environment

3.3.1. Project Management

Software development is always a team effort and requires a great deal of coordination. Although tools such as PVCS do provide project team management in the form of controlling access to project files and the way different team members can modify different parts of the same project, there is no build in coordination and management of the project team.

Enhancements should be made in the area of control access and project coordination in the context of parallel development.

3.3.2. Communication Support

The distributed nature of the software projects requires efficient and powerful communication support. SCM tools should provide efficient communication channels among project team member as well as other related project team members. Related personnel should be informed of changes to configuration items. The underlying communication architecture should provide the necessary mechanism in support of software activities such as software integration testing.

Though quite a few second generation SCM tools provide simple communication support, few of them available commercially can meet all the requirements.

3.3.3. Integration with Development Environment

All SCM tools should seamlessly integrate with the development environments in support of software activities such as design, coding and testing. The integration should provide users with the transparency that eliminates the need for detailed knowledge of the SCM tool [28]. None of the tools available commercially can provide such kind of integration, e.g. the command co in RCS for checking out files should preferably be replaced by the command ci to assimilate the UNIX environment. The closest to providing this kind of integration is PVCS which has an Application Programming Interface (API) to enable developers to build interfaces which integrate PVCS services with the user application.
Little consideration has been paid to the manner SCM tools should be integrated with the development environment and tools.

3.4 Status Accounting

Status accounting is absent in many of the SCM tools. Good status accounting should track all activities on configuration items. It should inform users of the what, who and why on the item. It should provide flexible but informative reports for management reviews and development reviews. Tools such as SCCS offers simple status accounting information such as the release level, date and time of creation etc.

3.5 Configuration Audit

This is an area rarely implemented in any of the commercially available SCM tools. As most software projects involve end-user departments as well as developers who may be dispersed geographically, the authorization procedures can be extremely troublesome. Some form of workflow system should be in place to ensure proper procedures are followed.

3.6 Vendor Control

CVSII is one of the few systems that provide subcontractor (third party vendor source) tracking. The extensive usage of shareware in many software implementation projects has increased the importance of such control mechanism.

3.7 Visual Support

Visual search capability should be provided to pan and zoom across the project hierarchical structure. Users should be able to start from the top level with a more abstract view of the system. They can then navigate down to the subsystem level to retrieve the desired item. Wein [29] has discussed the perceptual issues relating to the benefits of visual support for version management.

Visual supports are general provided but with very limiting function.

3.8 Flexible Project Structuring

Configuration items for a software project normally come under the categories corresponding to different stages of the development life cycle:

1. Requirement and Design Specifications (documents)
2. Product
3. Plans
4. Tools & Libraries
5. Test & records

Category 1 should include all user workflow as well. The term product should include both source and object programs for the project (preferably with program logic). Category 3 includes plans such as the test plan, the installation plan, the implementation plan and quality plan. Tools are the tools used in the project. Libraries are the library functions included in the object programs. Test should include all the test cases and test sets used. Results of the tests should be properly recorded. All project structures should be designed in a flexible way to allow users to choose different combinations, e.g. for a programmer, the project structure should only contain 1, 2, 3 and 4.

System configuration should include all constituent items of the system. SCM tools should identify the correct version of items to be used as well as identifying the content (constituents) and the dependent relationships among each items. Another important function of identification is to identify the overall structure of the software system. This is important on three counts:

1. User can gain a better understanding of the system and the inter-dependency to facilitate development and maintenance,
2. The build process will run without error.
3. Whenever an item is changed, only those affected components need to be changed.

Both SCCS and RCS fail to provide management of the configuration as they solely rely on MAKE to compose the configuration. MAKE maintains dependency information of items in makefile. One of the disadvantages of MAKE is that if the operating environment does not record adequate timing information on items, MAKE cannot determine which file is out of date. In MAKE, the system is broken down into subsystems which stores in sub-directory under the system directory. On system construction, an activation file which stores all revisions is executed to retrieve items for composition.

Domain Software Engineering Environment (DSEE) [20] provides configuration management for system building. It includes a system model that describes the constituent items and their dependencies. It also defines a configuration thread that defined the version required for
each item. By combining the configuration thread with the system model, one can identified the desired system configuration. PVCS on the other hand supports a feature called footprinting. It is in a way like a bill of materials that stores a list of items for the whole system. It has a dependency scanning feature that provides an accurate representation among items. With NSE, the system structure is maintained by objects (mostly files) which contain attributes and link type to describe relationship among items [30]. Makefile information is used to track the dependencies between source and derived objects. In addition, NSE supports the logical system structure regardless of the physical location of items through the component concept [24].

4. Conclusion

Most of the existing SCM tools have been designed with mainframes or mid-range development environments and their most of their interfaces are operated in command text mode rather than user-friendly GUI mode. More recent developments such as Intersolv's PVCS have improved user interfaces but are limited to PC and UNIX platforms only. Too much emphasis has been placed on source code and, the problems of identification and documentation control are rarely addressed. An area of confusion is in the usage of the terminology revision, version and release. Very few tools have actually implemented release control. Although some form of status accounting has been implemented in most of the tools, very little emphasis has been placed on problem tracking in relation to project quality and progress assessment. Distributed processing support has so far concentrated to LAN environments and should be extended to multi-platform WAN environments. Almost as a rule, configuration audit is not implemented in these tools. There is also a general lack of vendor control. In order to make SCM tools more effective, the authors proposed the following features should be incorporated:

1. As a norm, source code control is the main thrust of current tools, identification and documentation control should be implemented to improve the effectiveness of these tools.
2. SCM tools should preferably be run on PC platforms with remote access to different hosts (multiple-platforms). User-friendly GUIs are readily supported in Windows environment. Security features should also implemented to protect files downloaded form the hosts.
3. Tools must have the ability to support parallel development in a WAN environment TCP/IP should be used to allow developers to perform parallel software development on Internet.
4. Efficient email or communication support should be a standard feature in the implementation of user change request control.
5. Status accounting should include problem tracking and progress review reporting.
6. For the longer term, configuration audit should also be incorporated to enhance the functionality of these tools.

Although research on software configuration management is maturing, the authors suggest that future research work can focus on the following area to lead to a new generation of SCM tools.

First, can a standardized and unique CM model be evolved instead of continuously proposing different model such as composition and transaction model? The standardized model should combine the merits offered by each CM model. It is a major challenge to SCM tool design. However, the first step for next generation SCM tool should incorporate multiple CM model together in harmony and try to establish a framework that offer the core concept and functionality for the new tool.

Second, should the new SCM tools have a more detailed representation of software architecture? Today's SCM tools only focus on the configuration items for building compiled and executable objects. For example, the system model and configuration thread in DSEE just concentrates on building the executables. In fact, a good SCM tool should represent all configuration items such as include sources and objects, documentation, test data and libraries that all are related to a software product architecture. For example a set of test data are used by a particular source program for unit testing. The relationship among different type of configuration items should be represented. Besides, a good SCM tool should also represent other important architectural information, such as data or control flow connections among the modules[31]. These information facilitate parallel development since a software system can easier to divide into several distinct sub-systems based on the inter-connection information of modules.

Third, should SCM be integrated with CASE tools so as to provide a unique user interface, maximize data sharing and hence minimize redundant data, and allow effective control and coordination on individuals? Obviously, it is a good news on the users' point of view. However, CASE/CM integration face many technical problems. For example, different SCM tools use different
CM models and different CASE tools use different methodologies. Is it necessary to make fundamental change to the implementation of CASE and CM tools that obviously is not welcome by most vendors? Wallnau try to address key issues of integration of CASE and CM from a third-party integrator’s perspective in [32].

Finally, with the rapid development of world-wide inter-networking, there is a trend that team development will be distributed. Team members will be physically located in different sites. Future SCM tools should focus on the support of a distributed and parallel development that are suitable for Internet and Wide Area Network (WAN). The SCM should have a good communication support and ensure synchronization of activities on different sites.

References


