An Overview of Software Configuration Management

AUTHORS

AGHA SALMAN HAIDER

(Lecturer Jazan University, Kingdom of Saudi Arabia)

EMAIL:aghasalmaanhaider@gmail.com

SYED NUSRAT

(RESEARCH SCHOLAR)

SHRI JJT UNIVERSITY

MOBILE: 9899127668

EMAIL:Nusrat.syed@gmail.com

ABSTRACT

Executing software configuration management (SCM) in an organization lifts a variety of integration troubles. Every software configuration management system understands an exact Software configuration management model. For an organization, adopting a precise software configuration management model through the acquirer of a software configuration management system means to acclimatize the software process such that the model fits in. Additionally, adopting a software configuration management model is enduring commitment as the current software configuration management model does not integrate well with each other. Other integration harms happen in the software development environment, where obtainable tools must be unlimited to access software configuration management items.

Introduction

Executing software configuration management (SCM) in an organization lifts a variety of integration troubles. Every software configuration management system understands an exact Software configuration management model. For an organization, adopting a precise software configuration management model through the acquirer of a software configuration management system means to acclimatize the software process such that the model fits in. Additionally, adopting a software configuration management model is enduring commitment as the current software configuration management model does not integrate well with each other. Other integration harms happen in the software development environment, where obtainable tools must be unlimited to access software configuration management items.

Appleton and Berczuk (2003) have explained that "software configuration management processes and tools hold up at least 2 classes of tasks in the expansion process, management and everyday software development [2]. The management connected tasks that software configuration management supports contain identification of product components and their versions, status accounting, change control procedures and review and audit. For everyday activities SCM helps them, as a developer with version control functions that permit them to trace the composition of versioned software products precisely as they are revised, maintain consistency between mutually dependent components, build complied code, executables and extra derived objects from their sources".

The separation between management and development behavior doesn't make a group of sense. The things developers do are essential for the management support tasks to be significant; they can't spot product components if there is no product to identify. At times the SCM process, chiefly the management hold up aspects seems to obstruct development work as conflicting to enhancing it. One reason for this is that SCM processes are often defined with the goals and need of management primary and ignore the day by day needs of the developers [1].

This purpose of this paper is to explains what software configuration management is, provides guidelines on how to do it, and defines in detail what a Software Configuration Management Plan should contain.

Purpose of software configuration management

Software configuration management is concerned with the management of all software objects created throughout the software life cycle. This comprises requirements definitions, implementations, test plans, software architectures, project plans, test data, etc. these are arranged into software configurations by means of a variety of kinds of associations such as hierarchies and dependencies. In accumulation to source objects, SCM deals with derived objects such as complied code and executables. The evolution of software system is tackled by version control [3].

Usually, a version represents some state of a developing object. Evolution may occur along special dimensions. Version beside the time dimension is called revisions, where a new revision is intended to reinstate a preceding one. In contrast, several variants may coexist at a given time. For instance, a software system required to run on many platforms. Thus SCM deals with both the product space and the version space. Software objects and their relationships comprise the product space; their versions are organized in the version space.

Steps to constitute software configuration management

- Initial working
- Baselining
- Change management
- Management of workspaces
- Configuration status accounting
- Configuration audit

Software configuration management is a set of management regulations that acts as a point for integration of all planning, tracking and implementation activities related to the management of software items. It is integral to software development and appropriate to all phases of a software project. Essentially, SCM offers a disciplined approach for control, modification and maintenance of items in a software project and ensures lifelong product integrity[5].

This confrontation arises because of cumbersome procedures and a lack of understanding of the harms that can result if there is no SCM. An effortless yet effective SCM process accompanied with appropriate tools, training and hand holding is typically required to effectively implement SCM. Software configuration management for great real systems is an unattractive task. For such systems large scale software configuration management systems are necessary. Little systems do not need software configuration management except a way to endorsement. The function of the software configuration management concerning revisions is to register and accumulate changes as well as to designate and to recognize revisions. Fundamentally, revision management is done by managing sequences. Since, some parts of a system the common base are suitable for all variants some parts are valid only for a number of variants and a few parts are valid only for one variant, the software configuration management has to allow the creation of some set of variants (Solvberg, Steinholtz and Bergman (1990)).

Use of SCM in problem resolution

Making changes in software during the course of its development is indispensable. Making change might be an easy task. However, tracking the changes made is one of the most challenging tasks encountered by any software development organizations since several developers work on a single module and have their own versions of the same module. Software configuration management is one of the most prominent techniques adapted by organizations today in managing software changes.

Software configuration management affords the means of keeping track of the product status, change incorporated and various remedial actions taken. As it is integral to software development and spreads across all life cycle phases, SCM wants to be planned at the start of the project as part of development planning [4]. SCM activities need to be audited and status of SCM actions needs to be reported. Regularly, developers are somewhat resistant to the discipline required in SCM, viewing this discipline as bureaucratic and as hampering their work.

Software configuration management tools

A software object is some type of particular, machine-readable document produce throughout the course of a project. The document must be stored online to be completely controllable by an SCM system. Instances of software objects are requirements documents, specifications, interface descriptions, design documents, test programs, program code, test data, test output, binary code, user manuals, or VLSI designs. Each software object has a exclusive identifier and a body enclosing the definite information [6].

A position of attributes connected with software objects and ability for connecting objects through a variety of relations are also needed. For instance, attributes record occasion of creation and previous read access, and relations connection objects to their revisions and variants. The positions of attributes and relations have to be extensible; afterward sections will bring in a basic position. They also require an ability to explain subclasses or subtypes of the common software object. For illustration, the subclass may attach the language in which the body is written, or the structure editor used to create the remains, or whether the object symbolizes an interface or an implementation. The subclass also describes the position of operations obtainable on objects of that class, such as configuring, compiling, printing, etc.

SCM is nowadays a fine recognized software engineering regulation and almost certainly more than 100 SCM tools are presently accessible. These envelop an extensive variety of functions conflicting in functionality, price and difficulty. Even though SCM was primary developed in the 1970s, it was not awaiting the delayed 1990s that there was a detonation of SCM tool accessibility. The query of which tool is the most excellent is immaterial. There is no best tool but there is the most suitable tool depending on the business objectives of the clients. A lot of easy tools include essential functions such as version management and CM, adequate enough for little expansion teams. Advanced tools supporting distributed development, change management and sophisticated building are used in huge endeavor. In any case a single SCM tool without continuous process support is not enough for its victorious consumption. Even the majority difficult tools must be included into the procedure with additional tools.

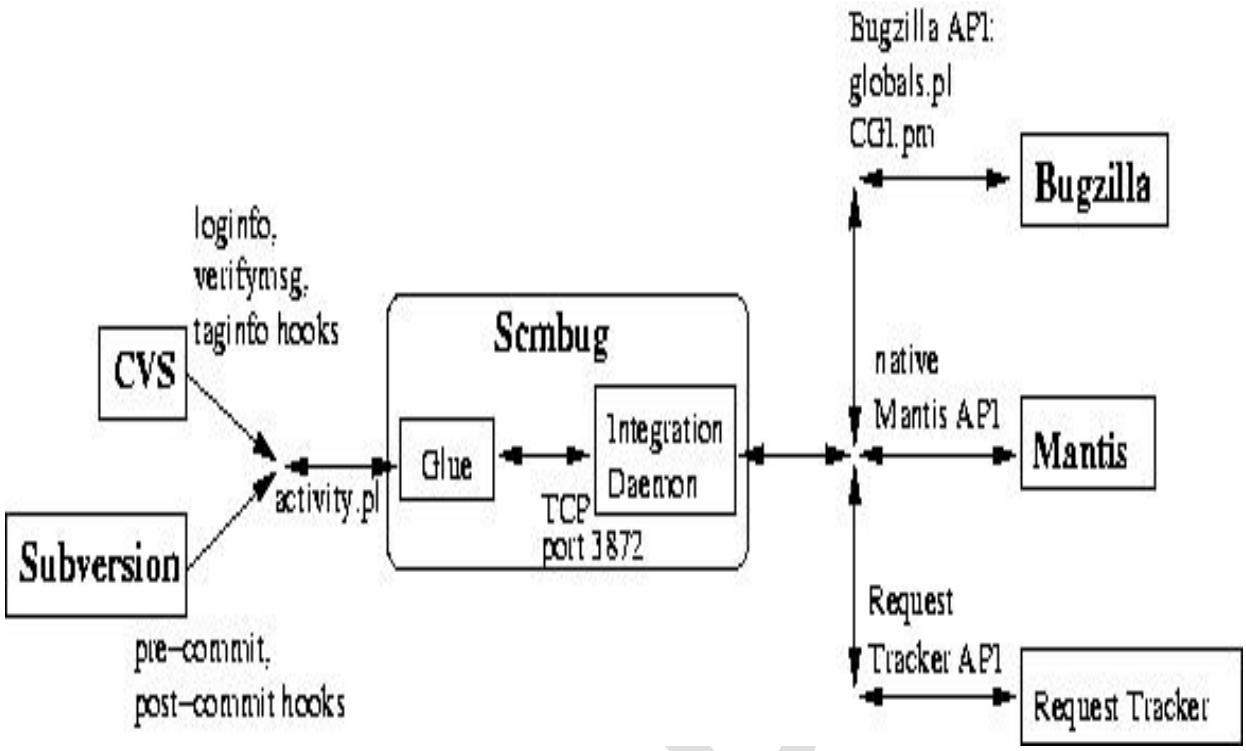

Figure 2.1: Software Configuration Management Tools

Source: Tan (2011), Informatics in Control, Automation and Robotics, Volume, Springer, p 663

Conclusions

Software configuration management is a discipline whose goal is to control changes to large software systems. The present state of the art is that managing and tracking the update of atomic objects via the check-in/edit/ check-out cycle is well understood. Reliably selecting versions and software manufacture are also well developed. The remaining paragraphs enumerate areas in need of further research.

Progress in accommodating and managing unavoidable inconsistencies in very large systems is still needed, as discussed by Schwanke and Kaiser [7]. An area presently under active investigation is version selection based on constraints and preferences. An area that is virtually untapped is representing and manipulating traceability relations, for instance the relations between specifications, their implementations, the associated documentation,

and the tests. MR-driven SCM and its integration with project management is also an underdeveloped area.
With the spread of workstations, the problem of manufacturing distributed applications has gained in importance. The difficulties in these applications involve interfacing multiple programming languages and operating systems, distributing configurations over a network of (possibly heterogeneous) computers, and initializing the processes and the communication paths. Two representative approaches are Matchmaker [4] and Agora [9].

Semantic modeling of SCM with semantic networks [7] or object-oriented programming languages is a topic worthy of further exploration. A semantic model of SCM would associate the various software object types with appropriate operations and organize the types into a class hierarchy with inheritance. The model would have to be extensible, for instance with new programming languages or configuration types. Another requirement is to accommodate versions of operations, for example versions of compilers. The benefits of semantic modeling are greater conceptual clarity, direct representation of the model for machine interpretation, more sophisticated operations and queries, and simplified implementation.

References

1. Sommerville (1996), Software configuration management, Springer, p 8.
2. Appleton and Berczuk (2003), Software configuration management patterns, Addison-Wesley Professional, 2003, p 8.
3. Hoek and Westfehtel (2003), Software configuration management, Springer, 2003, p 24.
4. Regan (2011), Introduction to Software Process Improvement, Springer, p 119.
5. Ramesh (2005), Managing Global Software Projects, Tata McGraw-Hill Education, p 87.
6. Chang, Handbook of software engineering & knowledge engineering, World Scientific, p 523.
7. Solvberg, Steinholtz and Bergman (1990), advanced information systems engineering, Springer, p 80.

8. Lori B. Alperin and Beverly I. Kedzierski. AI-based software maintenance.

In IEEE AI Applications Conference, February 1987

9. Roberto Bisiani, F. Alleva, F. Correrini, A. Florin, F. Lecouat, and R.Lerner. *The Agora Programming Environment.* Technical Report CMUCS-87-113, Carnegie-Mellon University, Department of Computer Science,March 1987