Evaluating Software Configuration Based Approaches to Support Change Maintenance of Granular Software Artefacts

Othman Mohd Yusop and Suhaimi Ibrahim

***Abstract*—Software does change and evolve during maintenance or creation another version of software. Adding up new requirements or features onto existing software will cause introduction of new elements inside software artefacts (e.g. class diagrams, source codes, use cases and etc). SCM based approaches are indeed needed to maintain ever grow artefacts along with maintaining artefacts relationships. Therefore SCM based approach has becoming a way to resolve the maintenance issues. This preliminary study has been carried out to look into possibility of how several SCM based approaches are been used in maintaining software artefacts specifically during maintenance phase.**

***Index Terms*—Software Configuration Management, software maintenance, software change, evolution control, lowest abstraction.**

I. Introduction

In software engineering, controlling and tracking changes tasks in software is done via management activity namely **Software Configuration Management** (SCM). Tasks revolve around configuration management practices are revision control, control item identification, change control, namely a few. These practices are used to control software system evolution. Software system consists of a multitude of individual components, which must fit together to ensure the functionality of the system as a whole. Software does evolve; this includes software testing artefacts during maintenance phase. Sometimes changes, improvising codes, creating stubs and testing drivers, managing various versions of testing documents are completely a tedious process to be handled. Without good or insufficient SCM, [1] typical problems may arise.

1) Developers/testers mutually overwrite each other's modifications in the source code or other documents, as simultaneous access to share files is not avoided.
2) Integration activities i.e. integration testing phase would impede.
3) Which version of test cases, test documents, etc would mismatch and over crossing on one another which lead to overwritten to unnecessary documents and versions.

There are several definitions regarding SCM. [2], [3] describes SCM as set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made. Software artefacts in [3] definition are defined as work products and in contrast, the software artefacts are stated as organization's products by [4] from the Institute of Configuration Management (ICM). [ICM'98] defines configuration management (CM) as the process of managing the full spectrum of an organization's products, facilities, and processes by managing all requirements, including changes, and assuring that the results conform to those requirements. [5] defines *SCM is the control of the* ***evolution of complex system*** and more pragmatically, it is the discipline that enable us to keep evolving software products under control, and thus contributes to satisfying quality and delay constraints.

Regardless whichever the definition used for SCM, the underlying activities will be always exactly same. SCM encompasses the everyday tasks within an organization, whether during software development or maintenance. Software changes are identified, controlled, and managed throughout project life cycle. [6] categorised SCM into configuration identification, change control, status accounting and audit. In this very research paper, writer did not elaborate further the SCM components highlighted by [6]. This paper is focused on approaches that embedded SCM element within their approaches and some reviews were made out of the studied approaches.

Numerous numbers of SCM-based approaches to combat the aforementioned problems were introduced. Component-Based [7], [8], UML-Based [9], [10], Unified Extensional Versioning Model-Based/UEVM [11], Object-Oriented SCM-Based [12], SCM-Based Test-Driven Development (TDD) are some example of SCM-based techniques and many other more have been introduced to resolve aroused problems in maintaining artefacts of software testing lifecycle. Some tools such as PHOCA, Odyssey-VCS are developed along the way to support the approaches.

This paper is organised as follows. Section II elaborates current SCM challenges and future directions. Section III discusses on SCM-based approaches. Next on section IV will talk about the results of this review followed by section V before concluded by section VI.

II. SCM Current Challenges and Future Directions

J. Estublier *et. al*. [13], had listed out some challenges pertaining to SCM and its future directions. Through literature studies, research candidate found out, the

Manuscript received September 19, 2011; revised September 29, 2011.

Othman Mohd Yusop and Suhaimi Ibrahim are with Dept. of Software Engineering, Advanced Informatics School (AIS), Universiti Teknologi Malaysia *International Campus*, Kuala Lumpur.

challenges cited from [5] works are still valid and being used as references till today i.e. [14]. Following are challenges which are still remain as issues in SCM research scope.

A. Functionality and Efficiency

The SCM core semantic like version control, repository or workspace support, are still functionally weak; due to lack of adaptability, usability and so on.

B. Product Data Management (PDM) versus SCM

Industrial product company needs to include significantly amount of software in its operation. They need tools to consistently ensure the harmonic balance during the. development of software product and its components. Computer Aided Domain Specific Environment (CADSE) is an example imitating the capabilities of PDM.

C. Process Support

Some SCM tools have process workflow support. This process workflow support acts as an assurance so that works becomes homogenous and less cluttered. By following the standard workflow process, redundancy and inconsistency can be lessening throughout the entire company workflow. A framework approach researched by [15] is a good sample of process workflow support.

D. Web Support

Through remote access, software artefacts can be possibly accessed through web facility, but remote access has some drawbacks i.e. missing of mechanism for managing remote concurrent engineering. Nevertheless remotely manage software artefacts over the web will trigger other problems and challenges. The problems and challenges will escalate to another level of difficulty i.e. geographically separated, different time zones, once distributed SCM gets introduced. [16] Studies support web based software configuration management.

E. Interoperability and Architecture

The services of SCM need to categorically segregate based on its functionalities and be able to propose a tailored solution, hence advanced SCM. Fundamental components or core components of SCM are always being part of the SCM tools and these components are interoperating, thus the core is the kernel of the SCM itself in term of architecturally and interoperability. [11] is an example of a fine-grained configuration management research studies besides [8], [17] which support SCM at component level respectively.

Based on these criteria, research candidate will investigate existing SCM based approaches against these challenges criteria. From this study, some result pertaining SCM perspective will be drawn. Prior to that, next section will elaborate a gist of each of the selected SCM based approaches.

III. Literature Review on SCM-Based Approaches

In this section will elaborate some literature review done on several SCM-based approaches.

A. Component-based SCM

There are studies embarked on fine-grained configuration management namely component. These studies [7], [8] posit that file structure configuration management is not dynamic enough to cope up with changes at granular level or lowest abstraction level of software artefacts namely classes, objects, etc. Additional component-based software engineering (CBSE) has been introduced and getting popular, traditional SCM could not handle CBSE approach [8], [18]. Hence quite a number of researches on this particular area have been conducted and some tools along the way have been produced to support their novel approaches i.e. Component-Interface Dependency Matrix/ CIDM [17] . The latter tool is capable to perform impact analysis besides macro software configuration management.

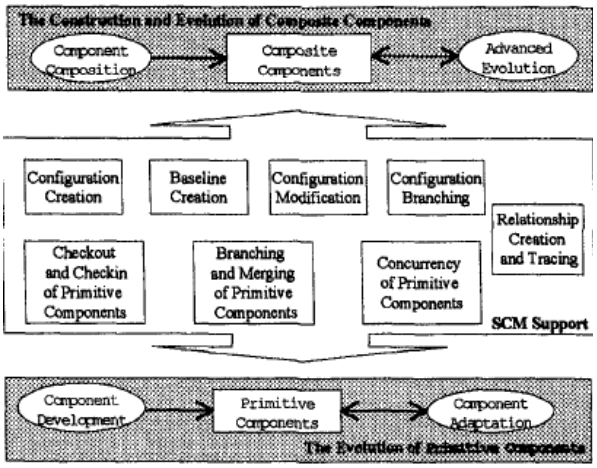

Fig. 1. Component-Based SCM Model

Fig. 1 above shows the component-based SCM model. The component-based SCM model was divided into two components; primitive component and composite component. Primitive component evolution will be maintained at the bottom of the model whereas the top part will handle composite components. Relationships will be treated as an object and will be put as part of object management.

There are even issues in CBSE, as quoted through [8], example: the issues of micro managing artefacts at granular level when the number of configuration items that need to be handle are bigger than conventional SCM which traditional SCM maintains a file (primitive file) rather than components and its links.

B. Unified Modeling Language (UML)-Model Based SCM

[7], [9], [10], consider models as first class artefacts. Models are considered important thus maintaining evolution of model just as important as maintaining source codes evolution. Model based SCM mainly supports Model-Driven Development (MDD™) that was created and proposed by Object Management Group (OMG). Models based version control make used of Unified Management Language (UML™), a graphic notational language created by the OMG. A tool called Odyssey-VCS inspired by SubVersion [19] is created to support model based SCM. Fig. 2 shows relationship between Odyssey-VCS versioning model. The model composed of five main classes namely version, configuration item, transaction, user and model element.

Each of these main classes are interconnected and give some meaning for instance each item from class ConfigurationItem has version. The version carries information such as time-stamping, branching, merging, description, etc.

In order to manage relationship among models (the versioning model and UML model), a higher level of model of abstraction is created as shown in Fig. 3 below.

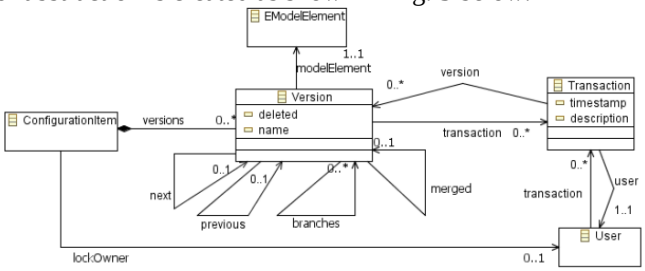

Fig. 2. The versioning model

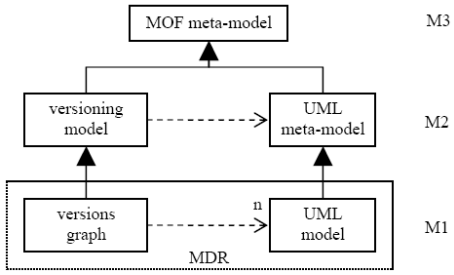

Fig. 3. Relationship between odyssey-VCS and UML

C. Unified Extensional Versioning Model Based

The structure used by version control systems is the same used by file systems, but in general the abstraction level made by software developers considers the file contents and its internal structure, including details as classes, methods, control blocks and others. Fine-grained version control tools can provide a more detailed version control. [20] However traditional tools and models provide very low flexibility and present high cost and impact of deployment in software development environments.

[20], proposed a model (Fig. 4) as a control element that has flexibility to refined versions, independent graphical interface plus a system called PHOCA to validate this research model. There are two features supported by this model; fine-grained and flexible version control. These features are achieved through creating a model bearing the requirements of refinement and flexibility that were desired, which involved the definition of flexibility in grammar and also the flexibility to use the tool. The conceptual model of the tool was inspired by Unified Extensional Versioning Model (UEVM).

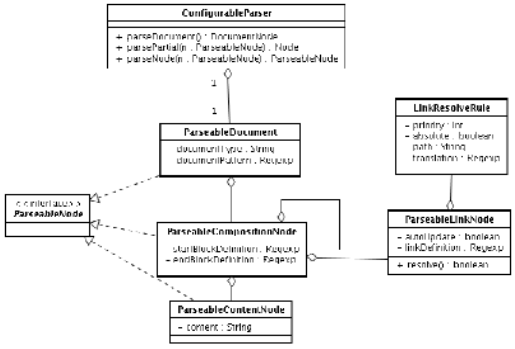

Fig. 4. Documents Meta Model

D. Object-oriented SCM Approach

[12] used SCM to improvise Software Architectural Traceability which they named it as Object-oriented Software Configuration Management (OO-SCM). The researcher found during the evolvement of software product, the design and the implementation code are bound to changes as well. Whenever changes are made upon changes, backtracking changes in design and implementation codes would produces greater troublesome. The classes and their relationships need to be managed as well, and from there these researchers found a necessity to introduced SCM features into ever growing software design. The approach was named as Molhado and shown as Fig. 5.

The goal of this approach is to be able to manage software artefacts on any domains at logical level. Logical level presents the lowest abstraction level of software artefacts i.e. classes, objects, relationships, etc. The idea was postulated into the approach which works by preserving the structural and the logical content of the software system into a data model. Structure versioning model is needed in order the control version of the software artefacts due to changes made.

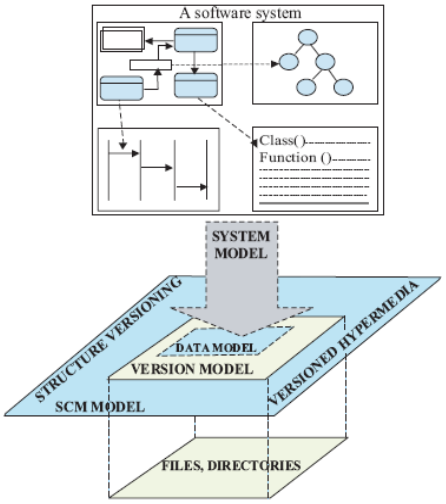

Fig. 5. Molhado Approach

E. SCM-Based Approach for Test-Driven Development (TDD)

One of the approaches that utilize software configuration management during testing development lifecycle is [21]. The research is done by incorporating configuration management phases or activities during Test Driven Development (TDD). Basically all steps and activities were recorded during the lifecycle and the approach was programmed as plug-in in IDE development tool.

TDD is a core test lifecycle for Extreme Programming software development process. The main purpose of TDD is to do quick testing before write up the program without jeopardizes the software design. Seizing failure results during the test is the purpose of having TDD. The more failures ones seizes, the better the program would become.

SCM used during TDD for this approach makes used of tree structure where a highest node represent a highest point of change and whereas the down line children are inheriting all information from their parents. The links among them would provide a backtracking means after changes or

refactoring were made to the system. All changes are categorically put into aggregating changes table as shown in Fig. 6 below.

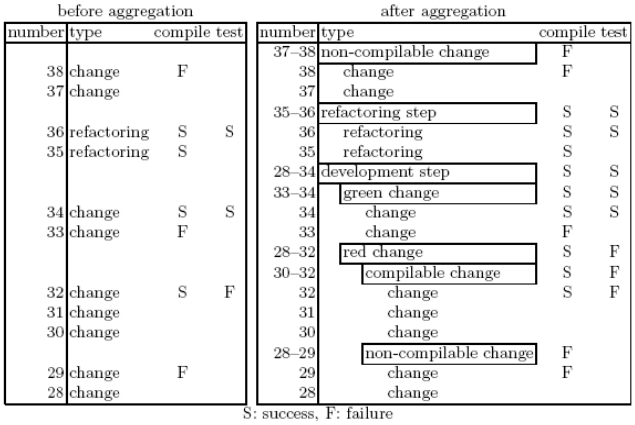

Fig. 6. Aggregating Changes

IV. Review Result

An evaluation was postulated into a result out of a review.

Table 2 shows the result of the review

TABLE II: Evaluation Result

Approaches vs Evaluation Criteria	Fine grained support	Lowest level of granularity	Supporting Tools	Traceability Support?	Phase in SDLC	Testing Phase Artefacts	Limitation (s)	Strength(s)	Answering SCM Challenges
Component-Based SCM	✓	✓	×	×	Maint.	×	Did not support Component at syntax level	Support large scale & primitive components	Functionality & Efficiency, Process Support, Interoperability & Arch.
Model-Based SCM	✓	✓	✓	✓	Analysis	×	Not stated	Support Meta Modeling	Functionality & Efficiency, Process Support, Interoperability & Arch.
UEVM-Based SCM	✓	✓	✓	✓	Analysis & Design	×	Require user intervention using supporting tool	A model and open source tool	Process Support, Functionality & Efficiency
OO-Based SCM	✓	✓	✓	✓	Analysis & Design	×	Avoid giving rules to composition-based arch. SCM	Able to control version of directory structure & evolution of arch. traceability	Interoperability & Arch., Process Support, Functionality & Efficiency, SCM
SCM for TDD	✓	✓	✓	✓	Maint.	×	Eclipse does not provide info on refactoring	Provide local change histories as plug-in	Process Support, Interoperability & Arch.

Table 1 shows a trend between conventional SCM and a component based SCM. The former technique explains the current practices applied during configuration management lifecycle. The elements involved are mostly high abstraction level of artefacts. Artefacts may compose of files, programs, etc. These artefacts may have relationships among them which were built up during the software development and tools may come to aid for artefacts archiving. On the other hand, component based SCM is more on handling evolution at the lowest abstraction of artefacts. These artefacts are fine-grained elements such as classes, objects, relationships, etc.

In table 2 shows Component-Based SCM (CB-SCM) that supports fine grained evolution control. Class, objects are sample of lowest level of granularity supported by CB-SCM. There are no supporting tools and traceability support though for CB-SCM. Model-Based SCM (MB-SCM), UEVM-Based SCM, OO-Based SCM and TDD-SCM do support evolution control on fine grained elements or at lowest abstraction level and there are supporting tools that built together along with the approaches. Each of the aforementioned approaches does incorporate traceability feature among the controlled elements. All of these approaches were used during some phases in software development lifecycle, except during testing phase. The writer tends to extend this evolution control capability to testing phase and work on the lowest level abstractions of testing artefacts.

Review is done by checking all the SCM-based approaches features against the criteria highlighted in section 2. Additional features were considered too, these features were captured based on similarity among these approaches. Some of these commonality features are: (1) fine-grained support, (2) lowest level of granularity, (3) supporting tools, (4) traceability support, (5) limitations and (6) Strength. Prior to that evaluation result, another additional outcome can be postulated out of the review as shown in table I. This table shows differences between traditional SCM which is heavily rely on file based relationships as opposed to latest trend in SCM which the latter is component based SCM:

TABLE I: Conventional SCM vs Component-Based SCM

Features	Conventional SCM	Component Based SCM
Level of granularity	*Coarse-grained*	*Fine-grained*
Primitive artefacts	*A File*	*A component and its links/connector*
Logical constituent	*Relationships among files*	*Relationships among components*

V. Threats to Validity

The evaluations are made based on the accepted published international papers and journals. Features and capabilities of the approaches are checked and justified against common features of configuration management such as evolution

control of the elements. Though there are no formal methods being used, the results were postulated based on our understanding and experiences in the field.

VI. Conclusion and Future Work

The review results presented here can be used as input data for further research work. There is possibility this SCM-based approaches get expanded to other phases such as testing phase artefacts. Software testing lifecycle alone has another complicated lifecycle that needs to properly maintained and versioned when it comes to its artefacts archieving.

References

[1] A. Spillner, T. Linz, and H. Schaefer, *Software Testing Foundations: A Study Guide for the Certified Tester Exam*. Rocky Nook, 2006.

[2] R. S. Pressman, *Software engineering: a practitioner's approach (2nd ed.)*. McGraw-Hill, Inc., 1986.

[3] R. S. Pressman and D. Ince, *Software Engineering: A Practitioner's Approach*. McGraw-Hill New York., 1997.

[4] V. C. Guess, *CMII of Business Process Infrastructure*. Hollis Publishing Company, 2002.

[5] J. Estublier, "Software configuration management: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, pp. 279-289, 2000.

[6] J. Keyes, *Software Configuration Management*. Auerbach Publications, 2004.

[7] L. Murta, H. Oliveira, C. Dantas, L. G. Lopes, and C. Werner, "Odyssey-SCM: An integrated software configuration management infrastructure for UML models," *Sci. Comput. Program.*, vol. 65, no. 3, pp. 249-274, 2007.

[8] H. Mei, L. Zhang, and F. Yang, "A software configuration management model for supporting component-based software development," *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 2, pp. 53-58, 2001.

[9] L. Murta, C. Corrêa, J. G. Prudêncio, and C. Werner, "Towards odyssey-VCS 2: improvements over a UML-based version control system," in *Proceedings of the 2008 international workshop on Comparison and versioning of software models*, pp. 25-30, 2008.

[10] L. G. Murta, A. Hoek, and C. M. Werner, "Continuous and automated evolution of architecture-to-implementation traceability links," *Automated Software Engg.*, vol. 15, no. 1, pp. 75-107, 2008.

[11] D. C. Junqueira, T. J. Bittar, and R. P. M. Fortes, "A fine-grained and flexible version control for software artifacts," in *Proceedings of the 26th annual ACM international conference on Design of communication*, pp. 185-192, 2008.

[12] T. Nguyen, E. Munson, and C. Thao, "Object-oriented configuration management technology can improve software architectural traceability," in *Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on*, pp. 86-93, 2005.

[13] J. Estublier et al., "Impact of software engineering research on the practice of software configuration management," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 4, pp. 383-430, 2005.

[14] J. Estublier et al., "Impact of the research community for the field of software configuration management," in *Proceedings of the 24th International Conference on Software Engineering*, pp. 643-644, 2002.

[15] Liang Ping and Li JianYang, "A Change-Oriented Conceptual Framework Of Software Configuration Management," in *Service Systems and Service Management, 2007 International Conference on*, pp. 1-4, 2007.

[16] D. YueHua, X. RiHua, and Y. Kui, "Design of VSS Software Configuration Management Database for WEB Application Project," in *Proceedings of the 2009 Pacific-Asia Conference on Circuits, Communications and Systems*, pp. 567-570, 2009.

[17] S. Huang, C. Tsai, and P. Huang, "Component-based software version management based on a Component-Interface Dependency Matrix," *J. Syst. Softw.*, vol. 82, no. 3, pp. 382-399, 2009.

[18] H. Mei, L. Zhang, and F. Yang, "A component-based software configuration management model and its supporting system," *Journal of Computer Science and Technology*, vol. 17, no. 4, pp. 432-441, Jul. 2002.

[19] Collins-Sussman, B. Fitzpatrick, B.W., and Pilato, *C.M. Version Control with Subversion*. O'Reilly, 2004.

[20] D. C. Junqueira, T. J. Bittar, and R. P. M. Fortes, "A fine-grained and flexible version control for software artifacts," in *Proceedings of the 26th annual ACM international conference on Design of communication*, pp. 185-192, 2008.

[21] T. Freese, "Towards Software Configuration Management for Test-Driven Development," in *Software Configuration Management*, 2003, pp. 143-150.

Othman Mohd Yusop received the Bachelor in Computer Science (1996), Master in Computer Science (1998), and currently doing his PhD in Computer Science. He is a lecturer attached to Dept. of Software Engineering, Advanced Informatics School (AIS), Universiti Teknologi Malaysia *International Campus*, Kuala Lumpur. He is an ISTQB certified tester (Certified Tester Foundation Level/CTFL). He is also an IEEE and ACM members. He has published several articles in international conferences and international journals such as the International Conference in Software and Information Engineering, International Conference on Digital Information and Communication Technology and its Applications, International Journal of New Computer Architectures and Their Application and International Journal of Information and Electronics Engineering. His research interests include software testing, software requirements engineering, software configuration management, software process improvement, secure software engineering.

Suhaimi Ibrahim received the Bachelor in Computer Science (1986), Master in Computer Science (1990), and PhD in Computer Science (2006). He is an Associate Professor attached to Dept. of Software Engineering, Advanced Informatics School (AIS), Universiti Teknologi Malaysia *International Campus*, Kuala Lumpur. He currently holds the post of Deputy Dean of AIS. He is an ISTQB certified tester and being appointed a board member of the Malaysian Software Testing Board (MSTB). He has published many articles in international conferences and international journals such as the International Journal of Web Services Practices, Journal of Computer Science, International Journal of Computational Science, Journal of Systems and Software, and Journal of Information and Software Technology. His research interests include software testing, requirements engineering, Web services, software process improvement, mobile and trusted computing.