INVENTEURS DU MONDE NUMÉRIQUE

Rocquencourt, September 3, 2012

Attestation of an accepted paper

CARI'12
African Conference on Research in Computer science and Applied Mathematics

I undersigned Eric Badouel, Secretary of the Permanent Committee of the African Conference on Research in Computer science and Applied Mathematics (CARI), certify that the paper :

Titel: Modelin software evolution through version control system
Autors: Hanene Cherait; Nora Bounour

has been accepted for presentation at the Conference CARI'12 to be held in Algier, Algeria, October 13 to 16, 2012. The paper will be published in the Proceedings of CARI'12 (http://www.cari-info.org) after the Conference.

The selection of the submissions is done on the basis of written evaluations of international experts, external to the Program Committee.

Eric Badouel
Secretary of the Permanent Committee – CARI
eric.badouel@inria.fr

Marie-Claude Sance-Ploucart
Inria - Direction of International Relations
Program Manager for Africa and Middle East
marie-claude.sance@inria.fr

[illegible]

Inria informatics mathematics
DEPT. OF INTERNATIONAL RELATIONS
Domaine de Voluceau - BP 105
Rocquencourt
78153 Le Chesnay Cedex France

Modeling Software Evolution through Version Control System

Hanene Cherait[1], Nora Bounour[2]

Department of Computer Science
Badji Mokhtar – Annaba University
Computer science research laboratory (LRI)
P.O. Box 12, 23000 Annaba ALGERIA
[1]hanene _cherait @yahoo.fr, [2]nora_bounour@yahoo.fr

ABSTRACT. Software engineering is concerned with the phenomenon of software evolution since software was developed. Real software systems require a continuous change to satisfy the user's new needs, and to avoid the degradation of software structure. To treat the evolution phenomenon, there are works that suggest software evolution models. Others have been interested by software evolution analysis. This last is mainly based on the information contained in versioning systems. But the evolutionary information they contain is incomplete and of low quality, hence limiting the scope of evolution research. In this paper, we analyze the interesting relationship between evolution models and version control systems to deduce a convenient way to integrate them in a single evolution model. We emphasize the works having aimed this integration. And, we illustrate this integration by our proposed model for version control system of aspect oriented software.

RÉSUMÉ. L'ingénierie du logiciel est concernée par le phénomène d'évolution du logiciel depuis que le logiciel a été développé. Les systèmes logiciels réels exigent un changement continu pour satisfaire les nouveaux besoins de l'utilisateur, et éviter la dégradation de la structure du logiciel. Pour traiter le phénomène de l'évolution, il y a des travaux qui suggèrent des modèles supportant l'évolution du logiciel. D'autres se sont intéressés à l'analyse de l'évolution du logiciel. Cette dernière est basée principalement sur l'information contenue dans les systèmes de contrôle de versions. Mais l'information évolutionnaire qu'ils contiennent est incomplète et de basse qualité, Ce qui limite l'étendue de recherche de l'évolution. Dans ce papier, nous analysons les relations intéressantes entre les modèles d'évolution et les systèmes de contrôle de version pour déduire une manière convenable pour intégrer les deux dans un seul modèle d'évolution. Nous mettons l'accent sur les travaux ayant visés cette intégration. Et, nous illustrerons cette dernière par notre modèle proposé pour le système de contrôle de version des programmes orientés aspect.

KEYWORDS: Evolution models, version control systems, change-based evolution, aspect oriented programming.
MOTS-CLÉS : Modèles d'évolution, systèmes de contrôle de versions, évolution basée changement, programmation orientée aspect.

1. Introduction

Software systems need to be changed in their lifetime, original requirements may change to consider the new ones or to support the system environment changes. These requirements for modifications have an impact on the overall software system. One of the preoccupations of the organizations is to evaluate this impact without implementing the changes. Evolution is then an important mechanism to replace these changes on the system and to guarantee its long life. The research effort spent in the software evolution topic shows the importance of the domain.

We conclude that the works in software evolution are focused on two main axes: evolution modeling and evolution analysis (Figure 1). The first one treats the evolution process, where different sources (source code, architecture, documentation…) are used to construct a software model. Then, evolution operations (changes) are applied on this model to guarantee that the software will evolved in a secure and coherent way using different mechanisms (change impact analysis, change propagation…etc).

The second axis is concerned with evolution analysis and more specifically history evolution analysis. Generally; the repository of version control systems is the basic source for understanding and analyzing software evolution. In recent years an extensive research has been carried out on exploiting the wealth of information residing in versioning repositories for purposes like reverse engineering or cost estimation. Analyzing version repositories can help to identify necessary changes, understand the impact of changes, and provide a facility to track the changes and to deduce logical relations between changed entities.

Although, these two axes are evolved separately but we believe that they converge to the same goal “software evolution”. This is why we suggest that the integration of them in a single process is a promising step toward an accurate and efficient evolution model. In this paper, we discuss the interesting relationship between evolution modeling and evolution analyzing to deduce a convenient way to integrate them in order to improve the software evolution process. We propose a new version control system produced by this integration. Our proposal is dedicated to aspect oriented paradigm. More attention is given to aspect oriented software evolution, because these systems are widespread used in nowadays as an efficient and modularized programming methodology.

The paper is organized as follows. In the next section, we explore the evolution analysis research area. Next, we discuss the limits of actual version control systems. Section 4 presents the change-based versioning as well as our proposal for aspect oriented software. Related works are surveyed in section 5. Finally, we conclude our discussion in section 6.

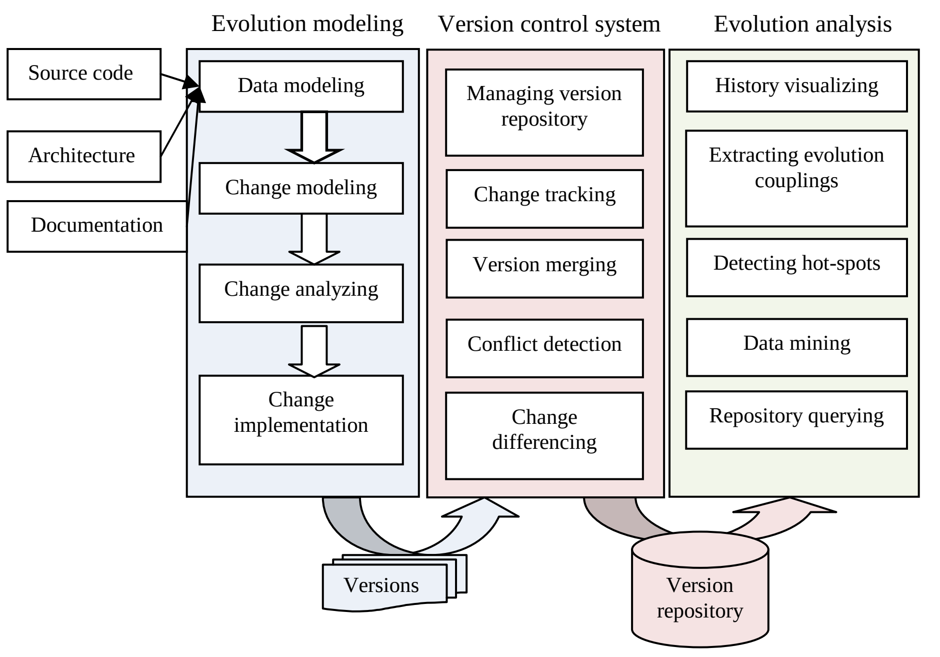

Figure 1. *Evolution Modeling and Evolution Analysis*

2. Evolution Analysis

To understand why software systems become less maintainable when changed continuously (program comprehension), and to reduce their maintenance costs eventually (reverse engineering); we have to investigate their version repositories. The research field of this investigation is known as *software evolution analysis* (Figure 1).

Software repositories contain a wealth of information about the software. The task is just to analyze them and uncover the information. Some works visualize the software evolution histories [5]. Other works focused on extracting logical dependencies "semantically coupled components may not structurally depend on each other" [9]. Such logical dependencies can be uncovered by analyzing the evolution history of a system. For example, in [8] the authors exploit historical data extracted from repositories and focus on change couplings. Ying et al [6] proposed a technique to determine the impact of changes based on association rules. In [9], the authors formalize logical coupling as a stochastic process using a Markov chain model. In [7] data mining is applied to version histories in order to guide programmers along related changes.

Other techniques are proposed to extract meta-information from the software repositories. For example in [3] the authors propose a model as a graph in which the different entities stored in the repository become vertices and their relationships become edges. They then define SCQL, a first order, and temporal logic based query language for source control repositories.

Based on our analysis of the field, we believe that "Indeed, actual versioning systems provide rich information for analyzing software evolution, but this information still coarse-grained and not accurate enough to perform an efficient evolution analysis". This is discussed in the next section.

3. Limits of Current Version Control Systems

The source code repository has a pivotal role for evolution analysis. However, the coarse-grained nature of the data stored by commit-based VCS often makes it challenging for a developer to analyze them [2]. In a previous study [4] it is showed that most versioning systems in use today are indeed losing a lot of information about the system they version. So, they are not plainly satisfactory for evolution research. There are two shortcomings which have major consequences, and are the cause of most of the other ones:

(1) Most VCSs are file-based, rather than entity-based. Romain Robbes and Michele Lanza claim that the commonly held vision of a software as a set of files, and its history as a set of versions does not accurately represent the phenomenon of software evolution "Software development is an incremental process more complex than simply writing lines of text" [4]. So, we can not follow the evolution of every entity in the software, and consequently, the evolution analysis is more difficult and not efficient enough for program comprehension or reverse engineering.

(2) Most VCSs are snapshot-based, not change-based. The program is frozen as a snapshot with a particular time stamp without recording the actual changes that happen in between two subsequent snapshots (recover only the end result of an evolution session). The time order of changes is lost, and it can not be perfectly derived. For understanding changes, the time order might be important. Moreover, the time order is useful for conflict detection and merging [11]. Groupings of changes to composite changes are lost. Refactoring operations e.g. cause many changes that can be grouped. This reduces the number of changes, and represents the change at a higher level of abstraction.

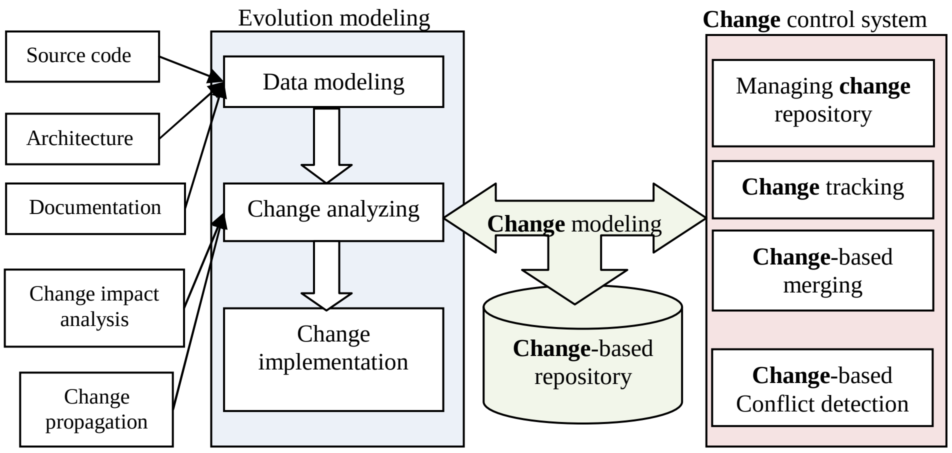

Figure 2. *Change-based evolution*

Deriving composite changes, e.g. to detect refactorings, is difficult and in some cases even impossible due to masking problems [11]. These disadvantages will reduce the ability of users to understand change. So the proposition of a new VCS, where the change is the first class entity can be a fundamental source for software evolution analysis. Many works prove that change-based evolution is more sufficient then state-based evolution ([2], [11]). In [2] the authors prove that for tasks that needed fine-grained change information, or in which the chronological order was important, change-based version control system outperformed the actual ones. And in [11] the authors describe the comparison between State-Based and Operation-based Change Tracking. To resume up, only a change-based VCS allows for effective research on evolution, since it provides all the required information. This VCS opens new ways for both developers and researchers to explore and evolve complex systems. The next section discusses this idea and presents or proposed version control system for aspect oriented software.

4. Change-based Version Control System

4.1 Principle

The history is today the center of evolution research. This is why; we believe that an accurate and efficient VCS is the key concept to improve software evolution. This last can be constructed by integrating the versioning mechanism in the evolution model of a software system. One effective way to perform this integration is by putting the source code change in the center of this process i.e. changes to the system must be stored directly in a change-based repository by the VCS (Figure 2).

The change-based VCS records the changes, while they occur, raising change to a first class concept. There is no need for differencing (Figure 2), since the changes are recorded and stored, and thus do not needs to be derived later on. It can preserve the time order in which the changes occurred. This is important information for understanding changes, but is also useful for other applications such as conflict detection and merging. The change-based VCS is more accurate, convenient and efficient then the actual ones. The rich change-based repository created by this VCS contains a more wealth of information about the software, so, it is a perfect source for a high quality evolution analysis.

4.2 Version Control System for Aspect Oriented Software

We have focused on the change-based evolution idea presented above to propose a version control system (change control system) for Aspect oriented software. We proposed a VCS dedicated for these software systems to gather the changes done on the system through the time (more details can be found in [1]). We suggest the use of the Algebraic Graph Rewriting as a support for the proposed VCS [13].

Our approach consists to model the source code as a colored graph; representing the different entities of the system and the relations between them. The evolution changes are formalized using rewrite rules on the system graph and stored in a rewriting rule based repository. In our approach, we do not view the history of a software system as a sequence of versions, but as the sum of changes which brought the system to its actual state. Our VCS follows this principle: (1) the change operations that serve to sail between the different versions of the software are considered as the rewrite rules on the source code graph. (2) we propose a rule-based repository as a versioning system: instead of recording the entire changed graph as a version, we only records the evolution sequences (the set of rewrite rules) on this graph, so a Version is: “sequence of rewriting rules applied to the aspect oriented software graph formulating a given evolution request” i.e. we can reproduce every version of the system by the execution of the sequence of rewrite rules associated.

The proposed version control system avoids the problems of actual version control systems “file-based, snapshot-based”. Our repository is change-based therefore a change can be described as one or more rewrite rules that change the program. In contrast to snapshot-based VCS our VCS does not keep track of different versions of an entity. Instead it captures all changes made to an entity as rewrite rules.

5. Related Work

Based on our analysis of the field, there is a very little work in this research area “change-based evolution”. The first work that treats this idea for object oriented software is the one of Romain Robbes and Michele Lanza [10]. They represent a state of a program as an abstract syntax tree (AST) of its source code. Then, changes to the program are represented as explicit change operations to its abstract syntax tree. Change operations are tree operations, such as addition or removal of nodes, and modifications of the properties of a node. Hattori and Lanza [12] extend Robbes’s change based software evolution model [10] into a multi developer context by modeling the evolution of a system as a set containing sequences of changes, where each sequence is produced by one developer. Thus, the evolution of a system comprises the combination of the sequences of changes produced by each individual.

Although, the idea of these works is similar to our proposal where the change is treated as a first class entity, but the use of the AST is not a good choice for software evolution (although it is more sufficient for software development). The AST captures the source code structure but it does not coverage it’s semantic, so the change repository is not sufficient for evolution analysis i.e. structural information is not enough for coupling detection, or data extraction,…etc. In contrast, our proposal is promising to better improve and accurate the change repository. We can capture structural as well as semantic information about the change (rewriting rules).

6. Conclusion

In this paper, we reviewed the state-of-the-art in software evolution research and we concluded that: To provide an accurate model for expressing software evolution process, we need to recognize the change as an explicit phenomenon and model it as a first class entity. This is performed by the integration of version control system in the software evolution model. Although, there is a little work for this axis of evolution research, but it is a promising way for best modeling and controlling software evolution process. We proposed a version control system for aspect oriented software, where the change is the first class entity.

Our proposal is based on the algebraic graph rewriting formalism which gives it a formal background and an automatic implementation method (employing graph rewrite tools). Our approach consists to model the source code as a colored graph; representing the different entities of the system and the relationships between them. The evolution requests are formalized using rewriting rules on the system graph. And, we proposed a rewriting rule-based versioning system to manage the evolution history eliminating the limits of the current versioning systems.

7. References

[1] H. Cherait and N. Bounour. “Toward a Version Control System for Aspect Oriented Software”. *In Proceeding of Model and Data Engineering (MEDI’11)*. Obidos, Portugal. LNCS 6918, pp. 110–121, September 28-30th 2011

[2] Lile Hattori, Marco D’Ambros, Michele Lanza and Mircea Lungu. “Software Evolution Comprehension: Replay to the Rescue”. *In Proceedings of IEEE 19th International Conference on Program Comprehension (ICPC),* pp. 161 – 170. 2011

[3] Abram James Hindle. “SCQL: A Formal Model and a Query Language for Source Control Repositories”. A Master Thesis. University of Victoria. 2005

[4] R. Robbes and M. Lanza., “Versioning systems for evolution research”, *In Proceedings of IWPSE 2005 (8th International Workshop on Principles of Software Evolution),* IEEE Computer Society, pp.155–164, 2005.

[5] Lucian Voinea, Alexandru Telea. “Visual data mining and analysis of software repositories”. *Computers & Graphics* : 31 pp. 410–428. 2007

[6] A. T. T. Ying, J. L. Wright, S. Abrams. “Source code that talks: an exploration of Eclipse task comments and their implication to repository mining”. *In Proceedings of International Workshop on Mining Software Repositories (MSR),* Saint Louis, Missouri, USA. 2005

[7] Zimmermann, T., Weissgerber, P., Diehl, S., and Zeller, A. “Mining version histories to guide software changes”. *IEEE Transactions on Software Engineering,* 31(6), 429–445. 2005

[8] Jacek Ratzinger, Michael Fischer, Harald Gall. “Improving Evolvability through Refactoring”. *In Proceedings of MSR*. Saint Louis, Missouri, USA. 2005

[9] Sunny Wong, Yuanfang Cai, and Michael Dalton. “Change Impact Analysis with Stochastic Dependencies”. Department of Computer Science, Drexel University, Technical Report DU-CS-10-07, October. 2010

[10] Lanza, M. and Robbes, R. “A Change-based Approach to Software Evolution”. *In Proceedings of ENTCS’07,* Volume 166, ISSN: 1571-0661, pp. 93-109. 2007

[11] Maximilian Koegel, Markus Herrmannsdoerfer, Jonas Helming, and Yang Li. “State-based vs. Operation-based Change Tracking”. *In Proceedings of MODELS ’09 MoDSE-MCCM Workshop,* Denver, USA, 2009.

[12] L. Hattori and M. Lanza. “Syde: A tool for collaborative software development”. *In Proceedings of ICSE 2010 (32nd ACM/IEEE Intl. Conf. on Software Engineering),* pp.235–238. 2010

[13] Ehrig H, Ehrig K, Prange U, Taentzer G. “Fundamentals of Algebraic Graph Transformation”. *EATCS Monographs in TCS*. 2005. Springer, ISBN 978-3-540-31187-4.