Comparative Evaluation of Change Propagation Approaches towards Resilient Software Evolution

Noraini Ibrahim
Software Engineering Dept.
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
noraini_ib@utm.my

Wan M Nasir Wan Kadir
Software Engineering Dept.
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
wnasir@utm.my

Safaai Deris
Software Engineering Dept.
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
safaai@utm.my

Abstract

Producing software that is adaptable to the rapid environmental changes and the dynamic nature of the business life-cycle is extensively becoming a topical issue in the software evolution. In this context, change propagation (CP) process is one of the critical parts in the software change management. Traditional strategies have projected more complex ways, resulting to substantial failures and risks. This paper presents an investigation and highlights on the desired criteria to provide better means to simplify the complicated CP tasks. The evaluation results may be used as a foundation in improving CP approaches that provide significant challenges in software evolution.

1. Introduction

Software evolution is unavoidable due to the dynamic nature of the business environment in software life-cycle [1]. Thus, producing software systems that are able to adapt themselves to rapid environmental and requirement changes has become a topical issue in the software engineering research [2, 3]. However, the task is most critical and costly in today's evolutionary software development [4, 5], time-consuming and error-prone to support developers throughout the evolution of large-scale projects and complex software systems [6]. Hence, failure in controlling changes may result to delays of the project schedule, as well as high development cost [7].

Focus on change propagation (CP) research arena continues to grow as software evolves. It is becoming a need in helping software engineers and maintainers to improve their productivity and quality of work. They require a better and efficient mechanism to maintain the associations and consistencies between the different types and granularity level of artifacts once the changes are implemented. Therefore, focus on the change propagation process is a very vital activity that needs to be discovered in helping software maintainers to avoid any omission in identifying and propagating critical change to interconnected artifacts [8].

Nevertheless, there is little work done to highlight and classify the current change propagation approaches previously. This paper is organised as follows: Section 2.0 provides an overview of CP, the CP definition, its process, and a brief description on the state-of-the-art of CP approaches. Section 3.0 discusses on the comparative evaluation approaches that consist of the framework criteria, the overall results of comparative evaluations and the critical discussion on the result. Section 4.0 presents the future trends for the CP research works and its conclusion. Finally, section 5.0 describes the whole summary of this paper.

2. Related Works

In general, all components are related and depend to each other in one software system. The relationship established is consistent until one of the links is broken due to the changes introduced to the software at any duration during its life-cycle. Such changes might appear in terms of adding, removing or updating the functionalities and capabilities of the software components. Therefore, when some parts of the software are changed and modified, many other parts of the software need to be identified and changed as well [9]. This is where the change propagation fits into an area of the software change. Previously, CP approaches have been widely studied under the umbrella term of impact analysis in software evolution. However, in this paper we will specifically focus on the CP techniques that are being used with regard to the change management process that supports software evolution as a whole.

978-0-7695-3372-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICSEA.2008.71

2.1. Change Propagation Definition.

Hassan and Holt define change propagation as the "***changes required to other entities of the software system to ensure the consistency of assumptions in a software system after a particular entity is changed***" [9] [4]. In other words, it is a process of actually carrying out a set of initial modifications to the software components, and to re-establish the system's consistency, by making a set of estimated consequent changes. [10]. From the definition, it describes twofold critical questions on the CP: (i) which and when the components need to be changed after the initial changes have been implemented, and (ii) how to better control and maintain the existing relationships and dependencies that are currently established between the consistent components.

2.2. Change Propagation Process

Previously in the manual CP process, it attempts to propagate the next changes to the related components based on set of proposed impacts during the impact analysis phase and consult advices from the guru. Apart from that, maintainers also will work based on their own knowledge and prior experience to identify the next components to be changed. The change is iteratively propagated for each suggested components until there is no more changes needed [4]. To address this manual process, Hassan and Holt propose the basic CP model that shows the steps in managing the change request such as adding new requirement, enhancement in current software or fixing the bugs [4]. In the following Figure 1, we illustrate the basic idea to automate the CP process adopted from Incremental Software Change approach proposed by Chen [11]. The grey highlighted text box depicts the step of which fundamental process of change propagation that should be taken into account.

2.3. State-of-the-Art of Change Propagation Approaches

In this section, we discuss four approaches on existing change propagation topics, although they vary on the focus for the evolutionary software development. We provide an overview to highlight the summary of the mechanism, properties of software artifacts under this study and any other supports that are being used for each change propagation approach.

2.3.1. Evolving Interoperation Graphs Approach

In the early days of change propagation research, Rajlich proposes an evolving interoperation graphs to model the change propagation for the evolution of component-based software [5, 12, 13]. The basis of this prior model is that each change consists of finer and smaller granularity of change propagation steps. The graph illustrates the relationships and dependencies among the software components. Rajlich suggests four change propagation strategies based on dependency graph rewriting that include top-down, random change-and-fix, strict change-and-fix and bottom-up [13]:

Tools such as Ripples [14], JTracker[15] and JRipples [16] were implemented to support change propagation based on this model. Ripples tool is concerned on the change implementation for C source code. JTracker and JRipples focus on detecting the secondary as well as an additional change for the affected classes in Java codes. However, all of the above said tools are still conferring with human intervention and also need maintainers' previous experiences to perform manual change propagation process. Therefore, handling the change is error-prone because there is a possibility to introduce unnecessary and unanticipated effects or risks to other components.

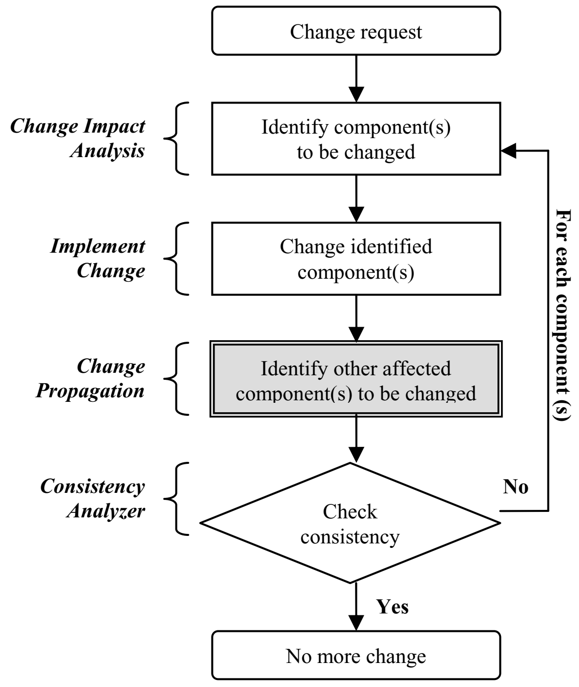

Figure 1. Basic model of a CP process

From the different preceding perspectives, Derulle et al. [10] propose Software Components Structural Model (SCSM) to perform change propagation on multi-language programs, heterogonous and distributed database applications that are based on the graph rewriting technique by Rajlich [13] to represent the software components and their relationships. They identified several numbers of problems when performing the existing change-and-fix algorithm and improved it by combining the algorithm with knowledge based system that is constraint by rules in an expert system.

2.3.2. Agent-Oriented Based Approach

Previously in software evolution, not much work in change propagation research focuses on agent-oriented software engineering, particularly in the development of agent designs. In the context of agent-oriented, once the new agent type is included, all other agents that are interrelated with the new agent type need to be modified too. From our best experiences upon exploration of the literature, only one work introduced by Dam et al. shows the agent-oriented approach dealing with propagating changes through design models [17]. The framework is based on the recognised Belief-Desire-Intention (BDI) agent architecture which is capable to support with consistency management in the context of the Prometheus methodology to design the agent based systems. Apart from that, the metamodel and the Object Constraints Language (OCL) are used to describe the rule forms or constraints for the automatic repair plan generation mechanism.

On the other hand, their work demonstrates that the proposed framework can also be applied to UML design for object-oriented methodologies. Additionally, they plan to further evaluate the framework with different and more complex case studies to measure its efficiency and scalability. Also, they present the extended agent-oriented mechanism for automatic repair plan generation to perform change propagation by fixing constraints inconsistencies when primary changes are made to a design models [18].

2.3.3. Historical Recorded Information Approach

Work done by Hamada and Adachi was among the earliest efforts that focuses on change propagation analysis [19]. They propose a method to support change propagation analysis by means of recorded software design rationale. The semantic and data models for the design process are established to provide software maintainers with an essential information needed by the change propagation analysis process to trace i) what are the design subject characteristics and ii) how the designer used the characteristics to design the software. They also develop a prototype system named DIG (Design Information Gathering) to implement the method and run the change propagation experiment on the requirement analysis. The result shows that the use of design process records is 15% less of the efforts required during the software modification and is 35% effective in terms of total cost savings during software development.

One of the current approaches that is based on recorded information to facilitate change propagation is Development Replay (DR) introduced by Hassan and Holt [9]. The DR approach uses the historical co-change information to estimate the effectiveness of not yet developed change propagation tools. [4,9]. The historical data (the state of the software project and the change sets) can be retrieved from the source control repository for each project handled by the software maintainers. The DR approach allows maintainers to highlight the limitation as well as to allow possible improvements of the studied tools. As a result, this can reduce the development's efforts and time especially for industrial studies. Hassan and Holt [9] claimed that this approach can assist researchers in propagating changes better than the previous simple static dependency information which are usually integrated and need substantial amount of human intervention. To assess the effectiveness of a heuristic or a tool in supporting the maintainers propagate changes, a metric of precision and recall has been used at the change set level.

2.3.4. Change Prediction Approach

AbdelMoez et al. propose a detailed architecture attribute, namely *Change Propagation Probability* that defines the likelihood or probability of a change from one or more architecture components and the consequences changes to other related components [20]. They use *Change Propagation Probability* to evaluate the design quality attributes of software architectures such as extensibility, maintainability and reusability. They also introduce *Change Propagation Coefficient (CPC) t*o store the information of a matrix in a single scalar, which demonstrates the likelihood of architecture to avoid its components from propagating changes to each other.

Additionally, they implement the change propagation matrix into Software Architectures Change Propagation Tool (SACPT) to display the Change Propagation Probability [21]. Clarkson et al. [22] use Design Structure Matrices (DSMs) to develop a mathematical models to predict the risk of change propagation in terms of likelihood and impact of change in complex design. Their work is slightly different to existing change propagation approaches because their intention is to analyse the changes execution processes by customising the design in an engineering product i.e. helicopters. The developed Change Prediction Method (CPM) is needed first to identify the previous sources of change propagation instances that have potentials to be affected and high likelihood to occur again. Secondly, the cost can be estimated by capturing the impact of change using the CPM approach. The captured likelihood and impact relationships are then used to determine the potential propagation graph as well as to produce the product risk matrix.

3. Comparative Evaluation of Change Propagation Approaches

This section describes the evaluation framework criteria in relation to the current change propagation approaches. We present the classification for the fundamental criteria discussed in the literature to develop a comparative evaluation framework to highlight and discover the critical attributes in performing change propagation without considering the domain of the changes sources. Moreover, the framework comprises criteria that are supported by most of the discussed approaches, as well as the criteria that must be underlined as desirable factors by the software maintainers.

3.1 The Evaluation Framework Criteria

In this first mechanism criterion, aspects related to the technique, algorithm or metrics support, and automation are discussed. Then, properties of software artifacts under study such as type, granularity, dependency relationships and change flow are also evaluated. The last criterion to be discussed below is the type of other support that is needed in performing change propagation in terms of visualisation, prioritisation, notification, consistency checking and log history or versioning system supports.

As a whole, the following criteria considered in the evaluation framework have been classified in three main elements: the mechanism, properties of software artifacts under this study and any other supports that are being used for each change propagation approach.

Framework Criteria	**Brief Explanations**
Mechanism	
Technique	Does the approach apply any specific techniques during the change propagation process? i.e. analysis of the best propagation path in terms of time consuming and efficiency measurement [22]
Algorithm or **metric**	Does the approach explicitly use any particular metric or algorithm to implement the change propagation process?
Automation	Does the approach provide any support of full automated process or partially automated where it still needs a manual and human intervention (semi-integrated) to perform the task?
Software artifacts property	
Type of software artifacts	What type of software artifact or software life cycle objects (SLO) and work products are being addressed when the change propagation process took place? This is because different type of artifacts will influence the type of change support mechanisms that will be required. [23]
Granularity	What extent is the level of high fine-grained and low coarse-grained granularity for each type of artifacts being defined? [24, 25]
Dependency relationships	Does the approach apply change propagation process within the same phase of artifacts (Horizontal or intra-phase: i.e. links from requirement to requirement) or across the different level of artifacts (Vertical or inter phase: i.e. links forward from requirement to design or links backward from design to requirement)? Or does the approach apply any strategies to maintain the defined traceability links [25-27]
Change Flow	What type of change flows are covered by the approach? Direct from change sources to the primary affected artifacts (i.e. a change in artifact A will affect artifact B) or indirect to secondary and hidden dependent artifacts (a change in artifact A will indirectly affect artifact C, as artifact B is dependant on artifact C)? [12], [28]
Other supports	
Visualisation	Does the approach provide any support to visualise the affected artifacts during change propagation process? Or any other kind of visualisation for propagation path?
Change Prioritisation	Does the approach provide any technique to prioritise the sets of impacted artifacts that should be given a high priority in determining the consequences change during change propagation process? [22, 29]
Change Notification	Does the approach provide any mechanism to notify the stakeholders or the maintainers when there is any changes happened and that should be highlighted and during change propagation process? For instance; notify the person responsible when the requirement changes hands [25, 30]
Consistency checker	Does the approach provide any support to check the dependencies consistencies between the artifacts before the change being implemented and after the change propagation is done? For example, consistencies validation and checking on dependencies between changing requirements [5, 12, 31]
Log history or **Versioning System**	Does the approach provide any support to keep the log and history of all changes, status and other related information needed during change process? This is due to high possibility for sources that are likely to change in the future; which follows regular patterns [4, 9, 19]

Table 1. Change Propagation Framework Criteria

3.2 The Comparative Evaluation Results

We believe that the CP process is a very critical activity during change management in software evolution. At this point, it is important to comprehensively investigate all of the factors that are being influenced during CP process. In addition, the criteria that contribute to improve or degrade the ability in performing the propagation process must also be identified. Thus, we also believe that the best practice to plan for the software evolution is to better control and support the process of changing requirement. This is because, a large portion of total software lifecycle cost is devoted to introducing new requirements, and removing or modifying the existing requirements [32]. It is significant to focus on requirement as important sources of changes from the initial stage of software development [8]. Looking at the recent works on change propagation, we realise that most of the efforts are expended and the issue is addressed from low and downstream level artifacts such as code and design. [9, 12, 15], but not much focus on high level like requirements [8, 25, 33]. The reason is because low level artifacts are more concrete and informative compared to high level artifacts that are normally expressed in an abstract manner [34].

Maintaining the existing consistent traceability and dependencies links of the software artifacts effectively is the essence of change propagation problem. Hence, software engineers and maintainers need a proficient mechanism to preserve the consistent relationships between the components after changes have been performed.

Another topical issue is how to systematically develop a better process to simplify the rigorous CP jobs according to the needs in each specific change requests situation. Nevertheless, the accurate prediction of the CP process provides a significant challenge [35] because the goal is to maximise efficiency by minimising error when selecting the next affected components from the impacted set. Eventually, this can assist them from any omission in identifying and propagating critical change to interconnected artifacts. Therefore, the work to be developed in this area must not only be able to predict the affected set correctly, but also propagating changes must be easily done without expected period time. It is an increasing need to help software engineers and maintainers to improve their productivity and quality of work in software development lifecycle. Thus, we believe that the automated CP strategies will help to reduce human errors when predicting all anticipated components.

4. Conclusion and Summary

This paper provides the first step towards providing the state-of-the-art of current change propagation research. From the literature, it is clear that the current CP approaches have various functions and criteria. Hence, we present a classification of fundamental criteria to develop a comparative evaluation framework, which in turn helps the researchers to identify the strengths and weaknesses of current approaches, and consequently discover the opportunities of improvement to be addressed in our proposed approach.

5. Acknowledgement

This research is funded by the ***ScienceFund*** Grant by *Ministry of Science and Technology (MOSTI)* under ***Vote No.79265.*** The authors would like to thank *MOSTI* and *Universiti Teknologi Malaysia (UTM)* for their financial support, as well as to individuals for their involvement and invaluable feedbacks during the time that this research was being conducted.

6. References

[1] W. M. N. W. Kadir, "BROOD - Business Rule-Driven Object-Oriented Design," PhD Thesis, School of Informatics, University of Manchester, 2005

[2] S. L. Pfleeger and J. M.Atlee, *Software Engineering Theory and Practice*. New Jersey, USA: Pearson Prentice Hall, 2006.

[3] M. M. Lehman, "Software Evolution," *Encyclopedia of Software Engineering*, vol. 2, 2002, pp. 1507-13.

[4] A. E. Hassan and R. C. Holt, "Replaying development history to assess the effectiveness of change propagation tools," *Empirical Software Engineering*, vol. 11, no. 3, September, 2006, 2006, pp. 335-67.

[5] V. Rajlich, "Modeling software evolution by evolving interoperation graphs," *Annals of Software Engineering*, vol. 9, 2000, pp. 235-48.

[6] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," *Principles of Software Evolution, Eighth International Workshop on*, 2005, pp. 13-22.

[7] V. T. Rajlich and K. H. Bennett, "A Staged Model for the Software Life Cycle," *Computer*, vol. 33, no. 7, 2000, pp. 66-71.

[8] A. von Knethen, "Change-oriented requirements traceability. Support for evolution of embedded systems," *Software Maintenance, 2002. Proceedings. International Conference on*, 2002, pp. 482-85.

[9] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, 2004, pp. 284-93.

[10] L. Deruelle, M. Bouneffa, N. Melab, and H. Basson, "A change propagation model and platform for multi-databaseapplications," *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, 2001, pp. 42-51.

[11] K. Chen, "Incremental Software Change," PhD Thesis, Wayne State University, 2003

[12] V. Rajlich, "A Model and a Tool for Change Propagation in Software," *NSF Software Engineering and Language Program Summaries, Software Engineering Notes*, Jan. 2000, 2000, pp. 72.

[13] V. Rajlich, "A model for change propagation based on graph rewriting," in *International Conference on Software Maintenance (ICSM), IEEE Computer Society*, 1997.

[14] K. Chen and V. Rajich, "RIPPLES: tool for change in legacy software," *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, 2001, pp. 230-39.

[15] S. Gwizdala, Y. Jiang, and V. Rajlich, "JTracker-a tool for change propagation in Java," in *European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press*, 2003.

[16] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: A Tool for Program Comprehension during Incremental Change," *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*, 2005, pp. 149-52.

[17] K. H. Dam, M. Winikoff, and L. Padgham, "An agent-oriented approach to change propagation in software evolution," *Australian Software Engineering Conference (ASWEC'06)*, vol. 0, 2006, pp. 309-18.

[18] K. H. Dam and M. Winikoff, "Generation of Repair Plans for Change Propagation," *Eighth International Workshop on Agent Oriented Software Engineering*, 14 May 2007, 2007.

[19] M. Hamada and H. Adachi, "Recording software design processes for maintaining the software," in *Seventeenth Annual International, Computer Software and Applications Conference, (COMPSAC) 93*, 1993.

[20] W. Abdelmoez, M. Shereshevsky, R. Gunnalan, H. H. Ammar, B. Yu, S. Bogazzi, M. Korkmaz, and A. Mili, "Quantifying software architectures: an analysis of change propagation probabilities," *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005.

[21] W. AbdelMoez, M. Shereshevsky, R. Gunnalan, Y. Bo, S. Bogazzi, M. Korkmaz, A. Mili, and H. H. Ammar, "Software Architectures Change Propagation Tool (SACPT)," in *Proceedings of the 20th IEEE International Conference on Software Maintenance*, Chicago, 2004.

[22] P. J. Clarkson, C. Simons, and C. Eckert, "Predicting Change Propagation in Complex Design," *Journal of Mechanical Design(Transactions of the ASME)*, vol. 126, no. 5, 2004, pp. 788-97.

[23] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *Journal of Software Maintenance and Evolution: Research and Practice*, 2004.

[24] A. von Knethen and M. Grund, "QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces," in *19th IEEE International Conference on Software Maintenance (ICSM) 2003*, 2003.

[25] M. J. Smith, R. G. Dewar, K. Kowalczykiewicz, and D. Weiss, "Towards Automated Change Propagation; the value of traceability," Technical Report School of Mathematical and Computer Sciences, Heriot-Watt University, 2005.

[26] K. Kowalczykiewicz and D. Weiss, "Traceability: Taming uncontrolled change in software development," *Proceedings of IV National Software Engineering Conference, Tarnowo Podgorne, Poland*, vol. 10, 2002.

[27] S. Ibrahim, "A Document Based Software Traceability to support Change Impact Analysis of Object-Oriented Software," PhD Thesis, Universiti Teknologi Malaysia, 2006

[28] P. Jönsson and M. Lindvall, "Impact Analysis," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds.: Springer, Berlin, Heidelberg, New York, 2005, pp. 26.

[29] I. Shaik, W. Abdelmoez, R. Gunnalan, M. Shereshevsky, A. Zeid, H. H. Ammar, A. Mili, and C. Fuhrman, "Change Propagation for Assessing Design Quality of Software Architectures," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 2005.

[30] T. Olsson and J. Grundy, "Supporting Traceability and Inconsistency Management between Software Artifacts," *Proceedings of the 2002 IASTED International Conference on Software Engineering and Applications (Boston, USA, 2002)*, 2002.

[31] J. Han, "Supporting Impact Analysis and Change Propagation in Software Engineering Environments," *Proceedings. In STEP97, London, England*, 1997, pp. 172-82.

[32] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: A roadmap," in *IEEE International Conference on Software engineering*, Dublin, Ireland, 2002.

[33] S. Lock and G. Kotonya, "An Integrated, Probabilistic Framework for Requirement Change Impact Analysis," *Australian Journal of Information Systems*, vol. 6, no. 2, 1999.

[34] S. Lock and G. Kotonya, "Requirement Level Change Management and Impact Analysis," *Cooperative Systems Engineering Group, Technical Report Ref: CSEG/21/1998*.

[35] V. Rajlich, "Changing the Paradigm of Software Engineering," *Communications of the ACM*, vol. 49, no. 8, 2006, pp. 67-70.