A Review for Improving Software Change using Traceability Model with Test Effort Estimation

Mazidah Mat Rejab, Suriayati Chuprat, Nurulhuda Firdaus Mohd Azmi

To Link this Article: http://dx.doi.org/10.6007/IJARBSS/v8-i4/4153 DOI:10.6007/IJARBSS/v8-i4/4153

Received: *25 Feb 2018,* ***Revised:*** *19 Mar 2018,* ***Accepted:*** *05 April 2018*

Published Online: 18 April 2018

In-Text Citation: (Rejab, Chuprat, & Azmi, 2018)

To Cite this Article: Rejab, M. M., Chuprat, S., & Azmi, N. F. M. (2018). A Review for Improving Software Change using Traceability Model with Test Effort Estimation. *International Journal of Academic Research in Business and Social Sciencest*, *8*(4), 1178–1188.

Vol. 8, No. 4, April 2018, Pg. 1178 – 1188

http://hrmars.com/index.php/pages/detail/IJARBSS | JOURNAL HOMEPAGE

A Review for Improving Software Change using Traceability Model with Test Effort Estimation

Mazidah Mat Rejab[1], Suriayati Chuprat[2], Nurulhuda Firdaus Mohd Azmi[3]

Advanced Informatics School (AIS), Universiti Teknologi Malaysia, 54100 Kuala Lumpur, Malaysia

Abstract

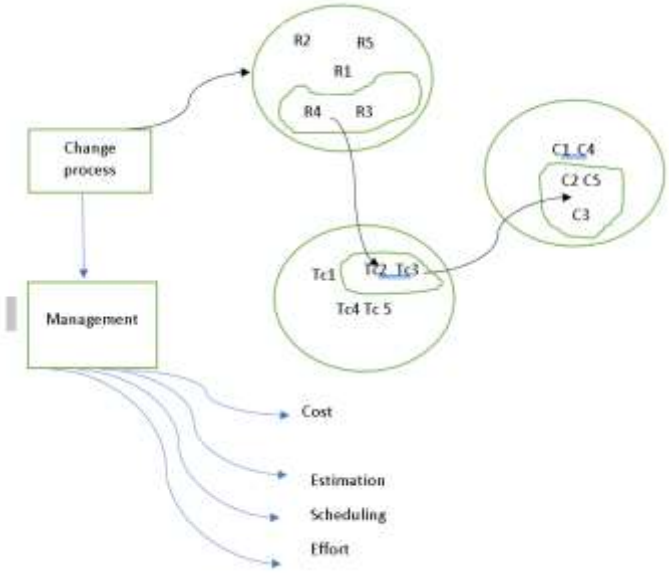
Maintaining a software system includes tasks such as fixing defects, adding new features, or modifying the software (software changes) to accommodate different environments. Then, the modified software system needs to be tested, to ensure the changes will not having any adverse effects on the previously validated code. Regression testing is one of the approaches which software tester used to test the software system. The traditional regression testing strategy was to repeat all the previous tests and retesting all the features of the program even for small modifications. For programming with thousand lines of codes (LOC), the cost of retesting the entire system is expensive if attempted after every change. This practice is becoming increasingly difficult because of the demand for testing the new functionalities and correcting errors with limited resources. Numerous techniques and tools have been proposed and developed to reduce the costs of regression testing and to aid regression testing processes, such as test suite reduction, test case prioritization, and test case done on the thresholds and weightings used in regression testing. However, there is still need to study on the software traceability model of coverage analysis in software changes during regression testing and test effort estimation on regression testing. Hence, this paper describes the proposal for improving software changes with hybrid traceability model and test effort estimation during regression testing. We will explain our proposed work including the problem background, the intended research objectives, literature review and plan for future implementation. This study is expected to contribute in developing hybrid traceability model for large software development project to support software changes during regression testing with test estimation approach and expected to reduce operational cost during the implementation on software maintenance. Also, it is hoped that an efficient and improve solution to regression testing can be realized, thus, gives the benefits to software testers and project manager manage the software maintenance task since it is a critical part in software project development

Keyword: Effort Estimation, Regression Testing, Software Traceability, Hybrid Traceability Model, Software Changes.

Introduction

The software application is present in every area in today's life. The small and large system is developed by using the software. Change is part of everyday life. Software changes after some time. In today's competitive atmosphere, brand new needs are arising, and existing requirements are altering swiftly. Changes are accomplished for various reasons, for example, including new elements, amending a few errors or for performing improvement. The changes are intrinsic in software and should be accepted as a fact of life. These changes are occurring very fast because of a competitive market (Lam et al 1999). Enhancing software is a universal necessity in business today as they encounter lots of need changes, prolonging software features and function, and including brand new modules. We cannot ignore the critically of software features and functions and including brand -new modules. We cannot overlook the critically of software changes because real software system changes and becomes more complex over time, [22.] The scope of software changes in this research is restricted to managing in the software itself. i.e., source code, design, test case and user requirement and change request. It will not cover issues related to changes in the software development environment, human resources, version control, project plan, and schedules. Suppose an artifact change has been requested by the client or an internal staff/ developer. This change can be either be in requirement, test cases or code, e.g., Classes, package, method, etc. A small change in requirement may affect some test cases and code and vice versa as shown in Figure 1 on the management side, can estimate cost, schedule time and human effort involved to cater this change.

Fig.1. Change Request Scenario



It is evident that one artifact change will affect the other artifact also. On the management of the software changes, it may affect the operational cost, estimation, schedule time and human effort. Software traceability is used to detect the modified software artefact during the change. Software change not only affects source code but also other artefacts like the design, test artifacts, and requirement. Requirement traceability offer support for numerous software

engineering tasks like requirement verification and validation., coverage analysis, impact analysis, and regression testing. On top of that, requirement traceability is the recognized component of software process improvement campaigns. Since the software evolution is inevitable, any traceability approach must consider signification affecting aspects so that it can reduce the evolution attempt Rochimah et al (2007). Regression testing is one of the methods which software tester used to test the software system. It tests the software after the changes to the software (for example; error fixes, or revise software) by testing the application again. The purpose of regression testing is to guarantee that a modification such as a bug solution, do not present new errors. Regression testing is carried out to go through most of the section of modified code and attain absolute test coverage.

The traditional regression testing strategy was to repeat all the previous tests and retesting all the features of the program even for small modifications. For programming with thousand lines of codes (LOC), the cost of retesting the entire system is expensive if attempted after every change. This practice is becoming increasingly difficult because of the demand for testing the new functionalities and correcting errors with limited resources. Numerous techniques and tools have been proposed and developed to reduce the costs of regression testing and to aid regression testing processes, such as test suite reduction, test case prioritization, and test case selection. Studies have also been done on the thresholds and weightings used in regression testing. However, there is still needed to study the software traceability model of coverage analysis in software changes during regression testing and test effort estimation on regression testing. Hence, the purpose of this research is to improve software changes with hybrid traceability model and test effort estimation during regression testing. From the model proposed it is expected to reduce operational cost during the implementation of software maintenance. This paper is organized as follows. In Section 2, the problem background related to the domain of the proposed study will be elaborate. This includes identification of the research questions concerning to the proposed work. Later in Section 3, the literature review regarding the problem domain identified is discussed. The proposed methodology for the proposed study is presented in Section 4. Finally, conclusion and future work towards the implementation of the proposed research are summarized in the last section.

Problem Background

Many people have realized that software and its applications will evolve as it adapted to changing environments, changing needs, new concepts and innovative technologies. The software will grow in the number of functions, components, and interfaces. Lehman [12] had described the previous modules might be extended for user beyond their original design. Subsequently, software modification and evolution are unavoidable. Much of the time spent on software maintenance is in modifying and retesting the software. Regression testing is a testing process which is applied after a program modification to detect errors introduced during change and to increase one's confidence that the modified programs till meets its specification. It involves exercising the revised program with some existing tests and new tests to re-establish assurance that the program will perform according to the (possibly modified) specifications. The problem of regression testing is not the same as the problem of testing, as most people have assumed. Regression testing involves testing the portion of the program affected by the modification, instead of the whole program. It is needed throughout the life of a product, once after every

change, while testing is done during the development of the product or its enhancements. The traditional regression testing strategy was to repeat all the previous tests and retesting all the features of the program even for small modifications. For programming-in-the-large, the cost of retesting the entire system is expensive if attempted after every change. This practice is becoming increasingly difficult because of the demand for testing the new functionalities and correcting errors with limited computer and human resources. An efficient solution to regression testing can bring significant benefits to software developers. Several studies have indicated that software maintenance and modification can require over 50 percent of the total life-cycle cost (Lientz, 2005). Testing the new functionalities and correcting errors with limited computer and human resources. An efficient solution to regression testing can bring significant benefits to software developers. Several studies have indicated that software maintenance and modification can require over 50 percent of the total life-cycle cost (Lientz, 2005)

For some projects, the maintenance cost can be higher. For example, for a given weapons system, about 25 percent of the software life-cycle costs are for development, and 75 percent for maintenance (G. Stuebing). Also, the increasing number of software applications being deployed may turn software maintenance into a significant subindustry by the close of the century, if not sooner. Numerous techniques and tools have been proposed and developed to reduce the costs of regression testing and to aid regression testing processes, such as test suite reduction, test case prioritization, and test case selection. Test suite reduction consists of removing unneeded or duplicated tests. Test selection is the process of identifying which regression tests apply to a given software change. Test case prioritization is the process of ordering those regression tests to either find faults earlier or to maximize the number of errors detected without running all the tests. Because the underlying motivation for regression testing techniques is to improve the cost-effectiveness of regression testing, research has also been done showing that proper thresholds and weightings must be used in regression testing techniques to improve economic benefits.

Software inevitably changes however well-conceived and well written it initially may be. Operational failures expose faults to be repaired. Mistaken and changed requirement cause the software to be reworked. New uses of old software yield new functionality not initially conceived in the requirements. The management of this change is critical to the continuing usefulness of the software. The new feature added to a system may be accommodated by the standard software development processes. According to Hartmann, regression testing attempts to revalidate the old function inherited from the old version. The new version should behave exactly like the past except where the partial operational requirement for new versions of the system. Much of this maintenance activity involves modifications to, and regression testing of, existing programs. Therefore, the importance of the regression testing process in software development cannot be overstated. There has been little work done in this significant area.

Numerous techniques and tools have been proposed and developed to reduce the costs of regression testing and to aid regression testing processes, such as test suite reduction, test case prioritization, and test case selection. Studies have also been done on the thresholds and weightings used in regression testing. However, there is still needed to study the software traceability model of coverage analysis in software changes during regression testing and test effort estimation on regression testing. Hence, we are proposing a study on improving software changes with hybrid traceability model and test effort estimation during regression testing. From

the model suggested it is expected to reduce operational cost during the implementation of software maintenance.

The hypothesis of the proposed study is as follow: "The hybrid of traceability model with test effort estimation in regression testing can improve the software changes with a reduction in operational cost." The main research questions derived from this study is: "How the hybrid of traceability model with test effort estimation in regression testing can improve the software changes with a reduction in operational cost?" From the main research question, sub-research questions are identified as below:

i. What are the characteristics of software traceability and test effort estimation in regression testing for improving the software changes?
ii. What are the types used approaches for supporting software changes in Regression testing?
iii. What are the components in enhancing software traceability model for reducing operational cost during regression testing?
iv. How to evaluate the improved software traceability model in reducing operational cost for a software project? The following section will describe preliminary literature review of the proposed study

Literature Review

A software change is inevitable at all stages of a software project. Change management will help the software Project manager direct and coordinate those changes. The only constant in software development is changing. From the original concept through phases of completion to maintenance updates, a software product is continually evolving. One single code change can significantly influence a wide range of software systems and their users.

For example, adding a new feature can spread defects in several modules, while changing an API method can improve the performance of all client programs. These changes determine whether the software meets its requirements and the project completes on time and within Budget. Studies by Aggarwal et. Al has investigated influential software changes which are;
i. Adding a new lock mechanism will influence not only specific to the module but rather on full system problem.
ii. Changing build configurations may affect the entire program.
iii. Improving performance in a particular environment may improve the of all applications using the library

Regression testing is part of traditional testing activities, which also comprise: test case generation, test execution, result evaluation, and coverage analysis. These events will cover the process of re-testing a software system after changes have been made to ensure that the new version of the system has retained the capabilities of the old version and that no new defects have been introduced. Regression testing is an essential activity, but it is also time-consuming and costly. Thus, regression testing should concentrate on those parts of the system that have been modified or which are affected by changes1. According to K. K Aggrawal & Y.Singh, they described the regression testing as: "Let *P* is the original software product, *P'* is the modified software product and *T* is the set test case to test *P*." A typical regression testing on modified

software process areas are as follow Select T'∈ T, a set of test case to execute on the modified software product P.'

[1] Test P' with T,' to verify modified software product's correctness concerning T.'
[2] If necessary, create T'', a set of new test cases to test P'.
[3] Test P' with new test case T'', to validate P' concerning T''.
[4] Create T'', a new test suite and test history for P' for T,' T'', T'''.

Over the past two-decade, a study by Rosero, R. H. et al had claimed that regression testing techniques give a commence relevant to the emergence of a new approach for developing the software. They have listed almost several methods in regression testing according to its criteria which are (Bennet, 2000) ; regression test minimization techniques,r egression test selection (RTS) techniques, regression test prioritization (RTP) techniques and regression test optimization (RTO) techniques. While automated code-based regression-testing support had mainly been studied and is supported by many tools, however, there is a lack of automatic support/studies on the feasibility of model-based regression testing (Cleland et al 2000)

Numerous techniques and tools have been proposed and developed to reduce the costs of regression testing and to aid regression testing processes, such as test suite reduction, test case prioritization, and test case selection. Studies have also been done on the thresholds and weightings used in regression testing. However, there is still needed to study the software traceability model of coverage analysis in software changes during regression testing and test effort estimation on regression testing. Many strategies utilize test case to code to determine code coverage nevertheless using both is boosting to gauge integrated hybrid coverage [6]. Test case coverage implies to the situation whereby a software application adjustment will lead to altering code at last. It results in generate coverage again after any modification. Therefore, coverage reconstruction has a crucial past in test coverage analysis.

The efficiency of code coverage analysis is used by a strategy. The usual coverage item utilized in different approaches includes statement, block, path, condition, loop, condition, modified condition/decision, method, class, package, design, and requirement. It could be obtained by rerunning the test case, traceability hyperlink existence measurement, and by regenerating traceability matrix. A useful approach in applying software traceability is through the implementation of traceability matrix. Traceability matrix is usually created by taking and placing identifiers of one document in the left column of the table and identifiers of the other document across the tops row of the traceability matrix.

A cross mark is placed at the intersection of the cells when an item in the left column is associated with an object over the top. This value specifies the mapping of these two items. A zero value is put in all the other non-related cross-section which indicates that there is no relationship between this column and row value. Traceability matrix may be used to view if the existing project demands are being complied with and assisting in the production of a request for proposal, numerous deliverable documents, and project plan activities.

The implementation of traceability matrix reveals that software traceability is useful to detect the affected software artifacts during the change. It is made use to connect software program artifacts (code, requirement and test case) in each of two means; code traceability (test case to code) and requirement traceability (requirement to test case). Traceability information provides as 'assistance' to the software developers or software maintainers to understand software dependencies among software artifacts in all software phases6. Traceability techniques

are mainly used between requirements and software test cases. Keeping the traceability information between software artifacts and requirements may be useful to evaluate the impact of change requests and help maintenance activities. Over the past decade, researchers have focused on specific areas of the traceability problem. Developing more sophisticated tooling, promoting strategic planning, applying information retrieval techniques capable of semi-automating the trace creation and maintenance process, developing new trace query languages and visualization techniques that use trace links, and applying traceability in specific domains such as Model Driven Development, product line systems, and agile project environments.

These issues and challenges have been addressed by Aranha and Borba[7] in their conceptual paper of the trends and future directions in software traceability. During the regression testing, test effort estimation will estimate of the testing length, effort, cost and schedule for a particular software test project in a proper environment for distinct methods, tools and techniques. The test effort is the establishment of the effort spends on test activity and the effort spends on debug activity9. Several approaches concerning test effort estimation have been proposed. For example, S. Nageshwaran proposed a simple mechanism to calculate software test effort estimation. This scheme made use of use case point to find out test effort. Later, D. Almeida et al. proposed an approach to estimation of test effort, with the help of software use cases. This method calculates test effort by the normal scenario of use case. Further, Z. Xiochun proposed estimating the size of test suit. Their work introduced the test case number execution complexity and tester defined as 3- dimensional vector to find out a test suit. The work discussed an approach which is based on before time estimation of test execution. Their proposed work summarizes that for testing of any program, tester predicts test case and then calculate execution complexity10. Recently, Kama et. al[13] had introduced new change effort estimation tool called as COCHCOMO (Constructive Change Cost Model). The model is claimed to be able to take into account the inconsistent states of software artifacts in its estimation process. To summarize,

Table 1 provide the summary of the existing studies conducted in the area of software changes particularly in regression testing with emphasize on coverage analysis techniques.

Table.1. Software Changes Based on Regression Testing Studies

Model & Author	Traceability Approach	Impact Analysis	Regression testing	Change Management	Test Effort Estimation
JavaCode Coverage Lingampally et al., (2007)	No	Yes	No	Yes (Before Change)	No
CATIA (Suhaimi (2006)	Yes	Yes	No	Yes	No
GRAYZER Faizah et al., (2012)	Yes	No	No	No	No
RequirementRTS Kapfhammer et al., (2008)	Yes	No	No	Yes	No
RTSDIF Mall R. et al., (2011)	No	No	Yes	Yes	No
HYCAT Shahid (2016)	Yes	Yes	Yes	Yes	No

Lam suggest that changing requirement are the main issues of the re-engineering and software maintenance activities. Also, a report published by the Standish Group International [21] which inquired 13,522 software project. Indicate that out of the surveyed project only 29 percent are successful, 18 percent is considered as" failed" and 53 percent are regarded as " suspected" and the main cause of the failed project is the requirement change[20]. This shows that most of the software maintenance activities concern on adaptive and completeness maintenance. For this intent, the organization must get to manage requirement adaptation as part of the border software evolution approach. Furthermore, Nurmuliani et. al states that 40% of requirement need rework during the software development projects.

Conclusion and Future Work

Several reasons for the difficulty of software change control can be found in the literature. These include loss of design knowledge, of the original requirement, accumulation of problem and change need. Furthermore, lack of design for change, time pressure, diversity of tools and information, poor images of changes function, poor maintainability of just releasing software, code decay, and verification across several product versions, and focusing on developing new software instead of managing the existing system. Therefore, it is hoped that with the proposed study on software traceability model for change impact analysis with test effort estimation during regression testing it will bring up the solution to aid the software testing team, project manager and another stakeholder in facing the request changes in software maintenance phase

Acknowledgment
The authors would like to thank Advanced Informatics School (AIS), Universiti Teknologi Malaysia (UTM) for the support of the resources.

Corresponding Author
Mazidah Mat Rejab. Advanced Informatic School (AIS), Universiti Teknologi Malaysia, 54100 Kuala Lumpur, Malaysia (mazidah3@live.utm.my)

References

Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R. (2007). Investigating effect of Design Metrics on Fault Proneness in Object-Oriented Systems. *Journal of Object Technology*, *6*(10), 127-141.

Aranha, E., & Borba, P. (2007, September). Test effort estimation models based on test specifications. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007* (pp. 67-71). IEEE.

Bennett, K. H., & Rajlich, V. T. (2000, May). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 73-87). ACM.

Cleland-Huang, J., Gotel, O. C., Hayes, H. J., Mäder, P., & Zisman, A. (2014, May). Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering* (pp. 55-69). ACM.

Kama, N., Basri, S., Asl, M. H., & Ibrahim, R. (2014). COCHCOMO: A Change Effort Estimation Tool for Software Development Phase. In *SoMeT* (pp. 1029-1045)

Kushwaha, D. S., & Misra, A. K. (2008). Software test effort estimation. *ACM SIGSOFT Software Engineering Notes*, *33*(3), 6.

Lam, W., & Shankararaman, V. (1999). Requirements change: a dissection of management issues. In *EUROMICRO Conference, 1999. Proceedings. 25th* (Vol. 2, pp. 244-251). IEEE.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*(9), 1060-1076.

Li, D., Li, L., Kim, D., Bissyandé, T. F., Lo, D., & Traon, Y. L. (2016). Watch out for this commit! A study of influential software changes. *arXiv preprint arXiv:1606.03266*.

Nageswaran, S. (2001). Test effort estimation using use case points. In *Quality Week* (Vol. 6, pp. 1-6).

Naslavsky, L., & Richardson, D. J. (2007). Using traceability to support model-based regression testing. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (pp. 567-570). ACM.

Nurmuliani, N., Zowghi, D., & Powell, S. (2004). Analysis of requirements volatility during software development life cycle. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian* (pp. 28-37). IEEE.

Ramler, R., Salomon, C., Buchgeher, G., & Lusser, M. (2017). Tool Support for Change-Based Regression Testing: An Industry Experience Report. In *International Conference on Software Quality* (pp. 133-152). Springer, Cham.

Rochimah, S., Kadir, W. M. W., & Abdullah, A. H. (2007). An evaluation of traceability approaches to support software evolution. In *Software Engineering Advances, 2007. ICSEA 2007. International Conference on* (pp. 19-19). IEEE.

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 years of software regression testing techniques—A survey. *International Journal of Software Engineering and Knowledge Engineering*, *26*(05), 675-689.

Shahid, M., & Ibrahim, S. (2016). Change impact analysis with a software traceability approach to support software maintenance. In *Applied Sciences and Technology (IBCAST), 2016 13th International Bhurban Conference on* (pp. 391-396). IEEE.

Sharma, A., & Kushwaha, D. S. (2012). Applying requirement based complexity for the estimation of software development and testing effort. *ACM SIGSOFT Software Engineering Notes*, *37*(1), 1-11.

Stuebing, H. G. (1984). A software engineering environment (SEE) for weapon system software. *IEEE transactions on software engineering*, (4), 384-397.

The Standish Group, Chaos, Standish Group Report, 2004.

Zhu, X., Zhou, B., Wang, F., Qu, Y., & Chen, L. (2008, September). Estimate test execution effort at an early stage: An empirical study. In *Cyberworlds, 2008 International Conference on* (pp. 195-200). IEEE.