A Novel Software Change Management Model

Mohd. Zia Ur Rehman
Department of Computer Science Engineering
IET, Mangalayatan University
Aligarh, India

ABSTRACT
It has been well accepted by the software professionals as well as researchers that software systems have to evolve themselves to survive successfully. Software evolution is a crucial activity for software organizations. The objective of this paper is to identify critical challenges and giving a proposal against those in the area of software change management. The paper focuses on the existing issues of software change management. Software change impact analysis, software change propagation and regression testing are the key steps in change management process.

Keywords
Software change management, software change impact analysis, software change propagation, regression testing, and software change complexity.

1. INTRODUCTION
Even today, software change management is a very challenging area for researchers. For about three decades, researchers have put their continued efforts in this direction and shared their contemplation. Change seems to be very simple when someone demands it, but the complexity of the task appears when it moves towards the implementation phase. Software maintenance and evolution is an expensive phase in the software development life cycle. The quality and demand of software can be judged by its maintainability.

As a change is requested, it is not the only issue as to where to make the change, but how the change is processed, is also an important consideration in order to maintain the quality of the software in terms of reliability, understandability, reusability and maintainability.

Before moving towards details of the change, just have a look at the following scenario. There is a complete well-furnished, multistoried building. After shifting in one of the flats of this building, the owner of the flat observes the need of some changes in the flat. A group of owners of different flats may observe the need of some changes in the common area, the building owner may himself think about some changes in the building in near future. Here, different categories of people generate the demand for change in the existing system within their limited premises as well as in the common area. These changes may be restricted by physical or financial constraints, or by some government rules. Hence, before implementing the requested change it is necessary to look and analyze the change request against such boundaries.

The similar scenario occurs with the software change. Different stakeholders and/ or users may request a change because of market competition or any other reason. The viewpoint of analyzing a single change is different for everyone. A change is always followed by multiple changes in the existing system. The most challenging activities during maintenance are: (I) to assess the impact of the primary requested changes and (II) secondary changes that are due as a result of primary requested changes. The ignorance of studying these secondary changes could make the system inconsistent and could generate new bugs; thus, it is essential that before actual implementation there should be a thorough impact analysis of proposed changes and the set of secondary changes.

The other dimension of change management is to assure the user that change is working correctly in the system without any side-effect in the existing system. Technically, we termed this process as regression testing. There are so many unfolded questions related with software change management. Answers to these may give efficient solutions to the software industrialists, developers and programmers regarding software change management.

2. STATE OF THE ART
Various authors propose software change process models. Olsen [6] presents a change management model that views the whole software development process including both development and maintenance phase. This model does not make a distinction between different types of changes and does not give any idea about, how different types of changes could be managed. According to Ghoshal [6], change management covers the change activities during both the software maintenance and development phases. Since the process is same for all changes, the change control board is also same for all changes, regardless whether the change request deals with a new functionality or with an error identified. An impact is the effect of one object on another. Impact analysis (IA) is used to determine the scope of change requests as a basis for resource planning, effort estimation and scheduling. Software change-impact analysis estimates the impacted elements and related documentation of the software, if the proposed software change is implemented. Arnold and Bohner [1] define a three-part conceptual framework to compare different impact analysis approaches and assess the strengths and weaknesses of individual approaches. Gethers et al. [2] proposes a framework for impact analysis in the paper based on the degree of automation and developer augmentation information, which is obtainable in a system maintenance scenario. The Pfleeger and Atlee [3] focus on the risks associated with the change and state, "Impact Analysis (IA) is the evaluation of many risks associated with the change, including estimates of the effects on resources, effort, and schedule". Kung et al. [4] describes an algorithm to identify the impacted parts of the system by comparing the original system and the modified version, and find the difference between these two systems. Sherriff and Williams [5] compute the impacted files based on textual similarities that have been retrieved from the previous Change Requests (CR).

Change propagation analysts analyze the flow of change and find the path required for change implementation. At every milestone, change transforms into other form. The deliverables of change propagation analysis is the detailed report, which consists of the inputs and outputs required for every milestone where change could move from. Dam [9, 10] develops a framework that provides a more effective and automated support for change propagation in design model that uses a Belief- Desire- Intention (BDI) platform to perform change propagation. The use case diagram consideration is absent in the proposed approach misses , which is the basic diagram for defining the deviation in system behavior from original to requested change in an abstract manner.

Chechik [11] proposes an approach that provides an automated technique for propagating changes between requirement and design model but the consistency between different models at different levels of abstraction is a major undertaking. Rajlich [12] gives a model for change propagation, which is based on graph rewriting. Weidlich [13] presents a novel approach of change propagation among business process models. His technique is based on the notion of a behavioral profile, which captures a set of dedicated behavioral aspects of a process model.

After implementing the requested changes in software system, the next step is to check the accuracy of the system, since during implementation of requested changes, a number of secondary changes are also made which may be a reason of performance degradation of the system. Test Case Prioritization is the process of scheduling test cases in an order to meet some performance goal. Agawral et al. [14] proposes a prioritization technique that achieves modified code coverage at the fastest rate possible. This work only concentrates on version-specific test case prioritization.

Huang et al. [15] proposes a method of cost-cognizant test case prioritization based on the use of historical records. In this approach, historical records are gathered from the latest regression testing and then a genetic algorithm is proposed to determine the most effective order of test cases. Seth [16] proposes an approach for prioritizing test cases using sequence diagram. However, the work is limited to a single diagram consideration of UML diagrams. There is also a need to consider the requirement of prioritization on the basis of requirement criticality.

There is various software change process models present in the literature, but every model handles all the changes by the single process. However, it is far from the real practices because a developer treats different types of change requests in different ways like urgent change demand bounded with very tight time span while changes arise due to market competition are tightly bounded with quality of the product time period of releasing is not the prime constraint. Following are some problems that are still uncovered.

- What is the impact on software complexity in the software change process?
- How much impact analysis is required?
- How much regression testing is necessary for assuring the system efficiency?
- What is the procedure to implement the change?
- What are the measurements to measure the impact of change on the existing system with the parameters like reliability, reusability, understandability and maintainability?

On the basis of literature survey, the research gaps mentioned above can be formulated as the following problem statement:

2.1 Problem Definition

Software change is an indispensable activity. A little change can affect the large part of the software, thus the managed way for analysis, implementation and testing of change is essential to maintain the software quality and functionality. The main objective of the research is to propose and implement a novel approach for managing the change. Focus of our work is to address the change related issues in UML based systems.

2.2 Research Objectives

The goal of this research work is to address the problems discussed above in an efficient manner. In order to achieve this goal, there is a need to resolve the following key issues that constitute research challenges for this work:

a. First, we need to develop a software change process model that includes change complexity computation.
b. Second, we need to design a formula for computing the software change complexity.
c. Third, we need to develop a technique, to determine how the proposed change would affect the existing system. This would involve impact analysis of proposed change to identify the components that would be influenced by the implementation of the proposed change.
d. Fourth, we need to design a methodology that will more precisely give the knowledge about secondary changes, with reference to the proposed change that covers the multiple design views [8] of the software design document, so that the updated and correct software artifacts will also be maintained.

Fifth, we need to develop a methodology for safely and precisely test the system after implementing

3. RESEARCH PRAPOSAL

In this research, it is assumed to find the solution of problems discussed in the previous section. The following points shortly give an idea, how these problems will be tackle:

- The complexity is one of the acute parameter for measuring the quality of the software. The complexity of the system increases as the system grows, but it should have some pattern. Here the complexity of change includes the parameters like the nature of change, scope of change, level of coupling, resource requirement for change urgency and level of change. Random complexity deviation is not good for the software health. It might be possible that a small change may affect more than one big change to the system. Thus, there exists a relation among software change impact, complexity deviation and system maintainability.
- There are different categories of a change: minor change, major change, urgent change, functional change, non-functional change, essential change. Putting same effort for all type of change in analyzing impact is not worthy. There is need to formulate some relation between severity of change and effort required in impact analysis.
- The complete system retesting after the implementation of proposed change is almost impossible. However, one cannot randomly decide the amount of testing that assures about software

efficiency and accuracy. Thus, there is a need of some method by which one can estimate the regression testing effort necessary for the software. Based on the criticality of the change, a portion of system gets impacted. The motive of the regression testing is to assure that the expected unimpacted part of the system remains unimpacted after implementing the proposed change.

- Change is a cyclic process that consists of many activities. For efficient implementation of a change, it is compulsory to follow some sequence of these activities. Somerville [7] proposes a change model, but this model does not consider the complexity of the change. Now, the question occur that how and when to analyze the complexity of the change. In this research, we will propose a novel change model.
- Most of the time the requested change comes in the category of functional change and we only put our focus on finding functional impact. To preserve the level of software quality, after imposing change in the existing system, is very challenging. Those changes can never be appreciated that result in degradation of quality of the system. The impact analysis of the quality parameters of the software system is the only way to analyze the effects of proposed change on the quality of the system. After filtering the research problems, the next section lists out the major objectives in order to handle software change management.

The following section discusses proposed software change process model.

4. PRAPOSED MODEL

Change is inevitable in the software development after the release of first version. Various authors propose software change process models. Ghoshal [6] presents a change management model that views the whole software development process including both development and maintenance phase. This model does not make a distinction between different types of change and does not give any idea about, how different types of changes have been managed. According to Ghoshal [6], change management covers the change activities during both the software maintenance and development phases. Since the process is same for all changes, the change control board is also same for all changes, regardless whether the change request deals with a new functionality or with an error identified. There is a need of a software change process model that handles the present issues.

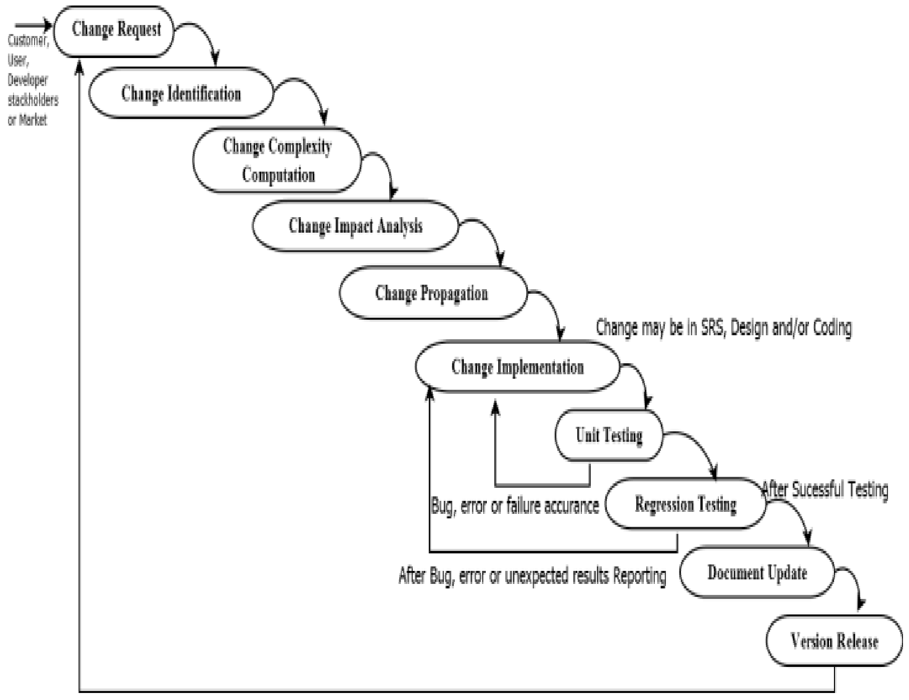
Customer, User, Developer stackholders or Market
Change Request
Change Identification
Change Complexity Computation
Change Impact Analysis
Change Propagation
Change may be in SRS, Design and/or Coding
Change Implementation
Unit Testing
Bug, error or failure accurance
Regression Testing
After Sucessful Testing
After Bug, error or unexpected results Reporting
Document Update
Version Release

Fig 1: Software Change Process Model

Table 1: Software Change Process Model Summery

Software Change Process Model Phase	Description	Deliverable
Change Request	Requests are accepted from different sources like customer, user, developer, market	Filled Change Request Form
Change Identification	Change categorization is done	Category wise requested change list
Change complexity computation	The complexity of change is computed	Requested change list with their complexity
Change Impact Analysis	Where the effect of change can arise	Impact set that contain impacted elements
Change Propagation	Identification of secondary changes required to implement requested change	Set of secondary changes for proposed change

Change Implementation	Actual change in existing system	Modified system
Unit Testing	It assures the functionality of change	Test result
Regression Testing	It assures that the existing functionality and quality of system does not get affected	Test result
Document Update	Change is made in the related documents like manual, data dictionary, training manual, installation guideline etc.	Modified documents
Version Release	New version is released	New version of system

The proposed software change process model consists of ten phases namely Change Request, Change Identification, Change Complexity Computation, Change Impact Analysis, Change Propagation, Change Implementation, Unit Testing, Regression Testing, Document Update, and Version Release. Again, it moves towards the initial stage as a change is requested. The functionality and deliverables of each phase can be summarized in the table 1.

In this model, Change Requests are taken by different sources like customer, user, developer and market demand. The next phase identifies the change; change could be functional change, non- functional change, architectural change, structural change, behavioral change, sequential change, parallel change etc. The handling of every type of change is not same. Here, work is limited to analyze only functional changes. After identifying the changes, the complexity of the identified changes is computed. In the next section, there is a detailed discussion about change complexity computation methodology. Aprna et at. [17] Proposes a fundamentally new approach that seeks a systematic solution to accept or reject the requested change. The next phase after complexity computation is the change impact analysis. This phase is the decision making phase. Stakeholders take the decision about acceptance or rejection of the proposed change based on various parameters like change complexity, effort required in implementing and testing of the proposed change, risk associated with proposed change, percentage of existing system influenced by the proposed change. The outcome of this phase is a set of accepted change. Now, the next activity is to fetch the set of secondary changes that occurred because of primary change at the different places of the existing system. This activity is carried by the change propagation phase. After finding secondary changes corresponding to the primary accepted change, change is actually implemented; the implementation means not only to make changes in the code but also making changes in system design. The unit-testing phase is carried out after completion of implementation phase. The implementation phase and unit testing phase makes a loop until the desired result are not assured by the unit-testing phase. The next phase after successful unit-testing, regression testing phase comes, it assures that the existing systems functionalities remain unchanged after implementation of requested changes. If some error, bug or failure is reported, process moves towards implementation by taking back step. Only after the successful regression testing, process comes at document update phase and the finally new version is released in the version release phase. In the proposed work, we will focus on change complexity computation, impact analysis, change propagation analysis and regression testing phase of the software change process model.

The next section is a conclusion of the paper.

5. CONCLUSIONS

In conclusion, the proposed research offers a fundamentally new approach that seeks a systematic solution to organize the software change management. Since, UML is an industry standard modeling language with a rich graphical notation, with comprehensive set of diagrams and elements. It is considered for the research work. In this work, a software change process model is proposed that includes the software change complexity computation before processing the change. The complexity computation gives a new dimension to the analysts for analyzing the requested change. In addition, the criteria of acceptance or rejection became strengthen than the earlier with the inclusion of change complexity. This research is expected to offer a more realistic solution for analyzing the software change impact of a proposed change on the existing system, closer values for software change propagation for making implementation process smooth and painless and regression testing that is more accurate.

6. REFERENCES

[1] R. S. Arnold and S. A. Bohner, “Impact Analysis - Towards A Framework for Comparison,” *Proceedings of the Conference on Software Maintenance*, Los Alamitos, CA, September 1993, pp. 292-301.

[2] Malcom Gethers, Huzefa Kagdi, Bogdan Dit, and Denys Poshyvanyk., "An adaptive approach to impact analysis from change requests to source code", In Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11), IEEE Computer Society, Washington, DC, USA, pp. 540-543, 2011.

[3] Pfleeger, S.L. and J.M. Atlee (2006)." Software Engineering Theory and Practice Upper Saddle River", New Jersey, USA, Prentice Hall.

[4] D. Kung, J. Gar, P. Hsia, F. Wen, Y. Togoshima, and C. Chen, “Change Impact Identification in Object-Oriented Software Maintenance,” *Proceedings of the Conference on Software Maintenance*, IEEE, Piscataway, NJ, USA pp.202-21, 1994.

[5] Mark Sherriff and Laurie Williams, "Empirical Software Change Impact Analysis using Singular Value Decomposition", In Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation (ICST '08), IEEE Computer Society, PP. 268-277, 2008

[6] S. M. Ghosh1, H. R. Sharma1, V. Mohabay, "Software change management – Technological dimension", International Journal of Smart Home, Vol. 5, No. 2, April, 2011

[7] Ian Sommerville, "Software Engineering, 9/E",ISBN-13:9780137035151 Publisher: Addison-Wesley

[8] IEEE Standard for Information Technology- System Design- Software Design Descriptions, IEEE Std 1016 ^{TM} 2009

[9] Hoa Khanh Dam and Michael Winikoff. 2011. An agent-oriented approach to change propagation in software maintenance. Autonomous Agents and Multi-Agent Systems 23, 3 (November 2011), 384-452.

[10] Hoa Khanh Dam and Michael Winikoff. 2010. Supporting change propagation in UML models. In Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM '10). IEEE Computer Society, Washington, DC, USA, 1-10.

[11] Chechik, M.; Lai, W.; Nejati, S.; Cabot, J.; Diskin, Z.; Easterbrook, S.; Sabetzadeh, M.; Salay, R.; , "Relationship-based change propagation: A case study," Modeling in Software Engineering, 2009. MISE '09. ICSE Workshop on , vol., no., pp.7-12, 17-18 May 2009

[12] Vaclav Rajlich. "A Model for Change Propagation Based on Graph Rewriting", In Proceedings of the International Conference on Software Maintenance (ICSM '97). IEEE Computer Society, Washington, DC, USA, 84-91.

[13] Matthias Weidlich, Mathias Weske, and Jan Mendling. ,"Change Propagation in Process Models Using Behavioural Profiles", In Proceedings of the 2009 IEEE International Conference on Services Computing (SCC '09). IEEE Computer Society, Washington, DC, USA, 33-40.

[14] K. K. Aggrawal, Yogesh Singh, and A. Kaur., "Code coverage based technique for prioritizing test cases for regression testing", SIGSOFT Softw. Eng. Notes 29, 5 (September 2004), 1-4.

[15] Yu-Chi Huang, Kuan-Li Peng, and Chin-Yu Huang.,"A history-based cost-cognizant test case prioritization technique in regression testing", J. Syst. Softw. 85, 3 (March 2012), 626-637.

[16] R. Seth, S. Anand, " prioritization of test cases scenario derived from uml diagrams", International Journal of Computer Application(0975-8887), Vol 46- No 12, May 2012.

[17] Aprna Tripathi, Dharmender Singh Kushwaha and Arun Kumar Misra. Article: Software Change Complexity: A New Dimension for Analyzing Requested Change. IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science 2012 ICRTITCS(7):5-10, February 2013