Understanding Change Prone Classes in Object Oriented Software

Deepa Godara
Research Scholar
Computer Science Engineering
Uttarakhand Technical University
Dehradun, India

R.K. Singh
Professor,
Electronics and Communication Engineering
Uttarakhand Technical University
Dehradun, India

ABSTRACT
Classes in Object Oriented Systems are continuously subjected to changes and defect prone. Predicting such classes is a key research area in the field of software engineering. It is important to identify such change prone classes and defect prone classes. Identifying change prone classes can help developers to build quality software on time. Considering all the above issues, this paper covers the following key issues: 1) identification of change prone classes using various approaches 2) How changes in one class affects multiple classes associated with it. 3) Study Dependency between classes and their effects.

Keywords
Change prone, class dependency, UML2.0 sequence diagram, UML2.0 class diagrams

1. INTRODUCTION
Software systems are continuously subjected to changes. Handling change is one of those fundamental problems in software engineering. Evolutionary development has been proposed as an efficient way to deal with risks such as new technology and imprecise or changing requirements [15]. Changes are made to add new features, to adapt to a new environment, to fix bugs or to re-factor the source code [4]. When adapting a system to new usage patterns or technologies, it is necessary to foresee what such adaptations of architectural design imply in terms of system quality [1]. Changes can be due to a variety of reasons such as enhancements, adaptation, perfective maintenance or fixing defects. Some parts of the software may be more prone to changes than others. Knowing which classes are change-prone can be very helpful; change-proneness may indicate specific underlying quality issues [3]. If a maintenance process can identify what parts of the software are change-prone then specific remedial actions can be taken. Thus, knowing where most changes are made over time can identify key change-prone classes, key change-prone interactions, and the evolution process can focus attention on them [3].

There are many reasons of project failures, some of them are: inaccurate understanding of end-user needs; inability to deal with changing requirements; software that is not easy to maintain or extend or late discovery of serious project flaws; overwhelming complexity; design and implementation; uncontrolled change propagation or insufficient testing [2]. Even a minor change can have considerable and unexpected effects on the system [11]. Classes that are more change prone in software require particular attention because they require effort and increase development and maintenance costs. Identifying and characterizing those classes can enable developers to focus preventive actions such as, peer-reviews, testing, inspections, and restructuring efforts on the classes with the similar characteristics in the future. As a result, developers can use their resources more efficiently and deliver higher quality products in a timely manner [9]. If faulty classes can be detected early in the development project's life cycle, mitigating actions can be taken, such as focused inspections Prediction models using design metrics can be used to identify faulty classes early on [13]. The accuracy of the predicted impact determines the accuracy of cost estimation and quality of project planning [14].

We believe that most of the software metrics evaluate the degree of object-orientation or measure static characteristics of the design, which are not always helpful in answering the question whether a specific design is good or not. When trying to answer such a question, an expert would assess the conformance of the design to well established rules of thumb, heuristics, and principles [10]. Behavioral Dependency Analysis (BDA) determines the extent to which the functionality of one system entity is dependent on other entities. Based on the source of information used to perform a BDA, we can divide the BDA techniques into three groups: code-based, execution-trace-based, and model-based. To derive behavioral dependency measures between two distributed objects, we perform a systematic analysis of messages exchanged between them in a set of sequence diagrams (SDs) For example, when an object sends a synchronous message to another object and waits for a reply, we define the former object to be behaviorally dependent on the latter [7].

UML is now widely accepted in the software engineering community as a common notational standard. It supports object-oriented designs which in turn encourage component reuse. It can be used to provide multiple views of the system under design [6]. The UML based design enabled us to apply formal verification and validation techniques [5]. The unified modeling language (UML) is a graphical language for visualizing, specifying, constructing, and documenting software-intensive systems. UML provides a standard way of writing system's blueprints, covering conceptual things, classes written in a specific programming language, database schemes and reusable software components [2]. UML has emerged as the software industry's dominant language and is already an Object Management Group (OMG) standard. It represents a collection of best engineering practices that have been proved successful in the modeling of large and complex systems. OMG is proposing the UML specification for international standardization for information technology [8]. As the use of object-oriented design and programming matures in industry, we observe that inheritance and polymorphism are used more frequently to improve internal reuse in a system and facilitate maintenance [12].

The remaining of the paper is categorized as follows: a short analysis of some of the literature works in the change proneness prediction methods is offered in Section 2. The inspiration for this study is specified in Section 3. Section 4 enlightens the short notes for the proposed change proneness

prediction methodology and the structure for the suggested methodology. The experimental results and presentation study discussions are given in Section 5. At last, the conclusions are summed up in Section 6.

2. RELATED WORK

In the course of the growth and preservation of object-oriented (OO) software, the data on the classes which are more prone to change is highly advantageous. Developers and maintainers are able to create further adaptable software by changing the segment of classes which are susceptible to modifications. Conventionally, nearly all change-proneness forecast has been investigated according to source codes. Nevertheless, change-proneness forecast in the initial stage of software growth can offer an easier method for evolving durable software by changing the existing plan or selecting substitute plans prior to execution. To tackle this requirement, Ah-Rim Han *et al.* [16] have offered an innovative and a systematic method for estimating the class dependency measure (BDM) which enables proper forecast of change-proneness in UML 2.0 brand. Ali R. Sharafat and Ladan Tahvildari [17] have come out with a novel method to forecast modifications in an object-oriented software mechanism. The key dilemma in software growth procedure is to evolve inaccuracy recognition to initial stages of the software life span. With this end in view, the Verification and Validation (V&V) of UML diagrams undertake a very significant function in identifying defects at the plan stage itself. It has a discrete relevance for software safety, where it is highly essential to spot safety faults before they can be subjugated. V. Lima *et al.* [18] have played a vital role in this regard by offering a formal V&V method for one of the most admired UML diagrams viz. sequence diagrams. A lion's share of research works has concentrated their attention on assessing the location of the utmost change-prone entities and the way of dissemination of the modification through a system. Mehdi Amoui *et al.* [19] have established that an awareness of probable time of occurrence of modifications will motivate managers and developers to design their preservation functions with superior proficiency. Premature detection of error prone and alteration prone classes enables the developers and experts to utilize their precious time and resources on these zones of software. Malan V. Gaikwad *et al.* [20] have the credit of introducing a novel a method of employing class hierarchy technique which is easily comprehend-able and executable. Recognizing the change-prone and inaccuracy prone classes earlier can help concentrating interest on these classes. Malan V. Gaikwad *et al.* [21] have intelligent focused on locating reliance of software that may be obtained by assessing the proneness of Object Oriented Software. Two major kinds of proneness were linked with OO software namely Fault Proneness and Change Proneness. Recognizing change-prone classes enables developers to devote further interest to classes with parallel traits in the future and thus investigation resources and time can be utilized more efficiently. Xiaoyan Zhu *et al.* [22] have gathered a group of static metrics and modification data at class level from an open-source software product, Datacrow. Moreover, Emanuel Giger *et al.* [24] have presented a paper for capturing the fine-grained Source Code Changes (SCC) and their semantics and also Ali R. Sharafat and Ladan Tahvildari [25] have proposed a novel method for the prediction of changes in object oriented software system, in which the quality aspects were qualified by the probability of change in each class.

3. CHANGE PRONE CLASSES

In this section we attempt to answer following questions:

What is change proneness? What are the advantages of finding change prone classes?

Change proneness is the probability that a particular part of the software would undergo change in future. Software changes can be due to: a) Addition of new features. b) To adapt to a new environment. c) To fix bugs. d) Refactor the source code. e) Due to enhancements, adaptation, perfective maintenance or fixing defects. Some parts of software are more prone changes than others.

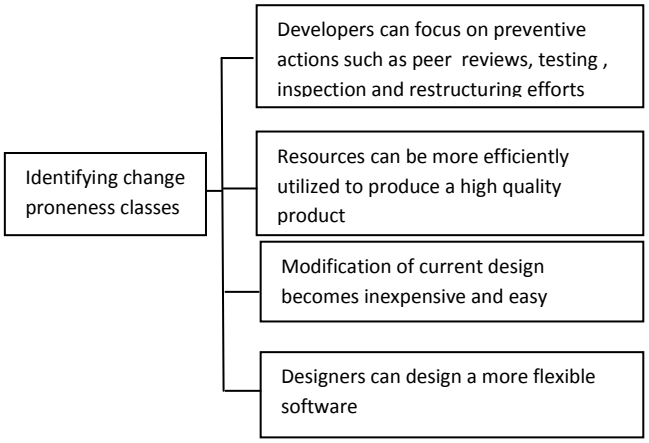

Fig. 1: Advantages of change prediction model

Finding such classes can be very useful: 1) Developers can focus preventive actions such as peer reviews, testing, inspection and restructuring efforts on the classes. 2) Resources can be more efficiently utilized for timely completion of project. 3) Developers can concentrate more on such classes to produce a high quality product. 4) A change prediction tool can improve maintenance and evolution tasks.5) If changes can be predicted at the earlier stages of software development when design models become available it becomes relatively easy and inexpensive to modify the current design. 6) Developers and Maintainers can make more flexible software by identifying change prone classes. 7) If change-prone classes can be predicted at the earlier phase in the software development life cycle, when the design models become available, quality problems related to design can be detected before implementing codes; current design can be modified or alternative designs can be chosen easily on design models.

Thus knowing where most changes are made over time can identify key change prone classes, key change prone interactions and evolution process can focus attention on them [3].

4. CHANGE PREDICTION ANALYSIS

Change prone classes can be analyzed by finding behavioral dependency using UML2.0 class diagrams such as class diagrams, sequence diagrams and interaction overview diagram. This section describes an overview of UML2.0 and its diagrams. UML 2.0 is totally a different dimension in the world of Unified Modeling Language. It is more complex and extensive in nature. UML is a modeling language used by system developers to specify, visualize, construct and document system.UML has become standard modeling language and it has expanded quite a bit since its inception and is applied to many different domains.UML has become

the de-facto standard for modeling software applications.UML2.0 is by far the largest UML specification, cleanest and most compact. It can be used for designing software, communicating software or business processes, capturing details about a system for requirements or analysis.UML attempts to bridge the gap between original idea for a piece of software and its implementation. One of the major motivations for the move from UML1.5 to UML2.0 was to add the ability for the modelers to capture more system behavior and increase tool automation. A relatively new technique called Model Driven Architecture (MDA) offers the potential to develop executable models that tools can link together and to raise the level of abstraction above traditional programming language. UML was designed to accommodate automated design tools, but it was not intended only for tools. UML 2.0 is distributed as four specifications: 1) Diagram Interchange Specification: It provides a way to share UML models between different modeling tools. 2) UML Infrastructure: It defines the fundamental low level, core, bottom concepts of UML. 3) UML Superstructure: It is the formal definition of elements of UML. 4) OCL Specification: It defines a simple language for writing constraints and expressions for elements in a UML model.

We will be using UML as a graphical language for visualizing, specifying, constructing and documenting software intensive systems. UML2.0 offers 13 diagrams. 1)Sequence diagram is a time dependent view of the interaction between objects to accomplish a behavioral goal of the system. The time sequence is similar to the earlier version of sequence diagram. An interaction may be designed at any level of abstraction within the system design, from subsystem interactions to instance level. It depicts the software in terms of a specific sequence of messages between objects. Here alt, opt and loop combined fragments enable modelling of complex control structures. 2) Communication diagram is a new name added in UML2.0. A Communication diagram is a structural view of the messaging between objects, taken from the Collaboration diagram concept of UML 1.4 and earlier versions. This can be defined as a modified version of collaboration diagram.3) Interaction Overview diagram is also a new addition in UML2.0. An Interaction Overview diagram describes a high-level view of a group of interactions combined into a logic sequence, including flow-control logic to navigate between the interactions.4) Timing diagram is also added in UML2.0. It is an optional diagram designed to specify the time constraints on messages sent and received in the course of an interaction.5) Class diagram provides structural information of classes and relationships between those classes.

A class can have two type of changes: internal changes and external changes.

Definition1: If a change is occurring due to modifications in a class itself is known as internal changes. These changes can be due to addition or deletion of attributes or any changes/modifications made to method declarations

Definition2: If a change is occurring by the changes propagated from other classes is referred to as external changes.

Internal changes can be predicted using source lines of code, number of parameters and number of fields whereas external changes can be determined by examining dependencies between pair of classes or objects in the system. Dependencies can be derived from UML2.0 diagrams such as sequence diagrams and Interaction overview diagram. From UML2.0 diagrams we can derive both structural and behavioural information, based on this we can derive behavioural dependency measurement. A class can affect other classes such that the other class get modified.

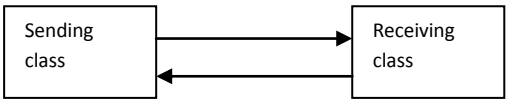

Fig 2 : Behavioral Dependency

If receiving class is modified it also causes the sending class to be modified as because modifying an object's class receiving a message may affect the object's class sending a message.

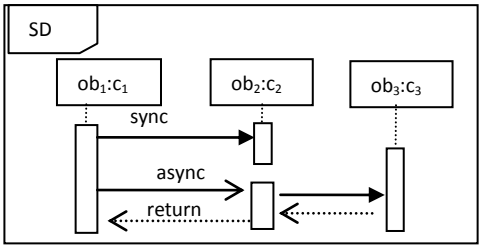

Fig 3 : Example of Sequence Diagrams

In Fig.3, the first message is sync i.e. a synchronous message (denoted by the solid arrowhead) complete with an implicit return message; the second message is async i.e. asynchronous (denoted by line arrowhead), and the third is the asynchronous return message (denoted by the dashed line). An object sending a message is behaviorally dependent on the object receiving a message. There can be two types of behavioral dependency: Explicit and Non explicit behavioral dependency. We assume that a system consists of objects ob_1, ob_2, ob_3, …..ob_n.

Definition3: If an object ob_1 needs to communicate with ob_2 by sending a sync message to ob_2 and receiving a reply from ob_2, it is said to have explicit behavioral dependency.

Definition4: If an object ob_1 needs to communicate with the object ob_r such that ob_1 needs some services of the object ob_2 by sending a sync message to ob_2 and ob_2 communicates with ob_r before replying to ob_1. Subsequently ob_r communicates with other objects before giving replying to ob_2 .This kind of dependency between objects ob_1 and ob_r is said to have non explicit behavioral dependency.

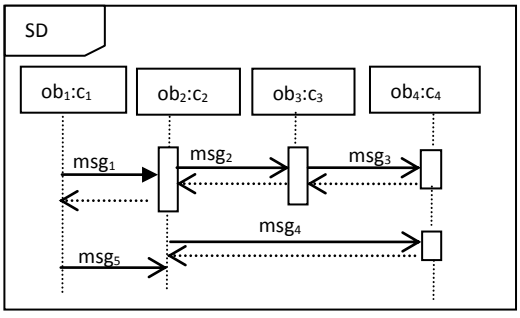

Fig 4 : Communication within Sequence Diagrams

In Fig4, object ob_1 communicates with object ob_2 by sending a sync msg_1 (synchronous message) to object ob_2 and receives a reply from it. Thus ob_1 exhibits explicit behavioral dependency with object ob_2 .On the other hand, object ob_1 communicates with object ob_3, before the object ob_1 receives a reply for the message msg_1 from object ob_2, object ob_2 communicates with ob_3 by sending message msg_2 .Since

object ob_1 does not wait for a reply from ob_2, object ob_2 does not exhibit any behavioral dependency.

5. CLASS DEPENDENCY

Class Dependency is one of the important features to predict the change proneness. In a source code, if one class gets changes, it also affects the other class. It can be found only through the dependencies between the classes or objects. The relationship between the sender object and receiver object is an example of the class dependency, while sending a message between two objects. The changes in the class of the receiver object also affect the class of the sender object. The inheritance and polymorphism are also taken into account, during the measurement of class dependency. The higher changes in class dependency indicate the possibility of more changes to happen.

Two kinds of class dependencies are:

Direct Class Dependency: - Consider two objects O_1 and O_2. If O_1 wants services to be get from O_2, then a synchronous message is sent to O_2 and O_1 waits for a reply that received from O_2. This kind of dependency is called as direct class dependency. This is denoted as, $O_1 \rightarrow O_2$.

Indirect Class Dependency: - Consider n objects $O_1, O_2, O_3 ..., O_n$. Indirect dependency between the objects O_1 and O_n is denoted as, $O_1 \Rightarrow O_n$, except $n = 2$ that represents direct class dependency $O_1 \rightarrow O_2$. Because the indirect class dependency is represented as $(O_1 \rightarrow O_2)\,\alpha\ (O_2 \rightarrow O_3)\,\alpha\ .....\,\alpha\ (O_{n-1} \rightarrow O_n)$.

Here, "α " indicates the External service request relation. For example, if an object O_1 needs a service from the object O_3 through O_2, then it is indicated as $(O_1 \rightarrow O_2)\,\alpha\ (O_2 \rightarrow O_3)$.

A synchronous message has the dependency between the sender and receiver objects, since the sender object depends on receiver object by waiting for the reply from the receiver object. It also indicates that the reply from the receiver object affects the sender object. But an asynchronous message does not have the dependency between the sender and receiver objects, since the sender object does not wait for the reply from the receiver object, its process continues. We have to compute the class dependency as a feature for our work, so we consider only the synchronous messages.

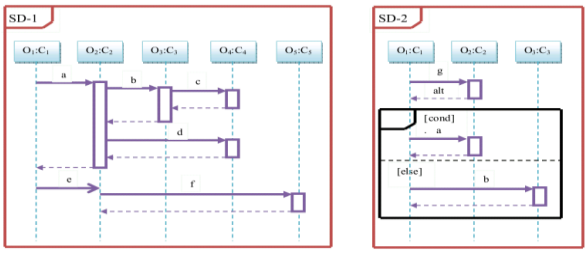

Fig. 5: General structure of Sequence Diagrams (a) SD-1 (b) SD-2

5.1 Measuring Class Dependency

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

To measure the class dependency from the source code, we generate a UML Sequence and Class diagram for the source codes. From this, we then measure the class dependency by using the following methods [16].

(i) Construct Dependency model of Object

(ii) Construct Dependency model of system

(iii) Form reachable path table

(iv) Calculate the weighted sum of reachable paths

(v) Calculate the Class Dependency

i) Dependency model of Object: messages are exchanged between instances of classes. Each message is a combination of three parts: reverse traceable message this is valid incase of indirect dependency, probabilistic execution (The probabilistic execution rate of a message is a probability of execution rate of a message in an alt combined fragment of a sequence diagram) and expected execution rate (probability of the execution rate of a sequence diagram)

ii) Dependency model of system: We need to find the dependency for the whole input application of source code. For this purpose, all these separated dependencies are combined together to build one big System Dependency Model to find the class dependency feature.

iii) Reachable path table: The paths between each pair of objects in the system dependency model are traversed from the source object to destination object and the paths are tabulated in a reachable path table. To find the reachable paths, traversal starts from a message incoming to the destination object to a message outgoing from the source object in reverse. These messages that are found via traversal are added in the reachable path table. If the source and destination objects have direct dependency between them, then the name of the incoming message to the destination object and the name of the outgoing message from the source object are equal. So, only one message name is included in the reachable path table for the direct dependency objects. But for the indirect dependency objects, the traversal from the incoming message to the destination object is carried out by iteratively substituting it with a backward navigable message and then we can reach the outgoing messages from the source object.

iv) weighted sum of reachable paths: Sum of reachable paths can be calculated as:

$$Sum = \sum \frac{1}{N} \times F_{PER} \times F_{EER}$$

where, N - Number of messages in the respective reachable path

F_{PER} - Probabilistic execution rate of first message in the reachable path

F_{EER} - Expected execution rate of first message in the reachable path

v) Class dependency: The Dependency feature for a particular class C_i, is calculated for getting the reachable path by summing the pair of instance of the corresponding class (C_i, C_j), where, $C_j, 1 \le j \le n,$ and n is the total number of classes.

6. PROPOSED CHANGE PRONENESS PREDICTION METHOD

Generally, the research on change-proneness prediction is made on the basis of "what" the researchers trying to predict and "how" they predict the changes. But most researchers are missed to find, "when" the changes are likely to be occurred. Most of the applications of object oriented software use complex inheritance relationship and polymorphism. Due to this reason, there has been less emphasis to capture the aspect of dynamic behaviors by the development of metrics.

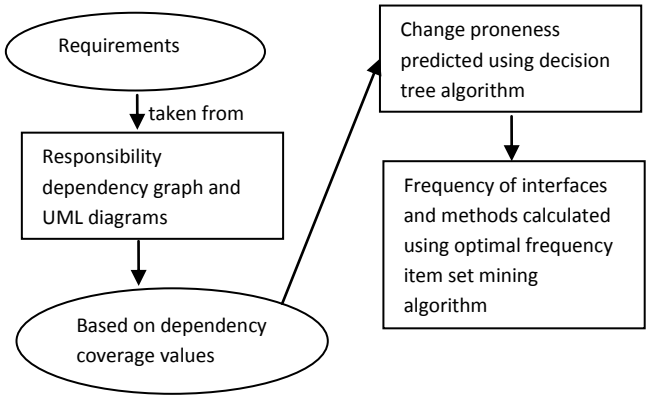

Fig 6 : Overview of efficient frequency based change proneness prediction model

Many of the existing metrics are still not explained about the substantial part of change prone classes to improve the change-proneness prediction models. Therefore, more number of information's is important to make an accurate change-proneness prediction model. In order to achieve maintenance, the frequency of changes in individual classes should be analyzed and also found the corresponding changes made in multiple classes. A change in one class affects another class also. It is needed to be analyzed the dependencies between the classes. This is one of the main issues in the prediction of change proneness. The existing methods concentrate only on behavioral dependency not the other factor which affects the change proneness. This leads to a complexity in predicting the change proneness of the system. In order to overcome these issues in our proposed model, we will use time, popularity, responsibility and dependency and other factors to predict the change proneness of the proposed system

7. CONCLUSION AND FUTURE WORK

In this paper, an overview of change proneness and the advantages of finding change prone classes at earlier stages is given. Then, we discussed behavioural dependencies, its types and how it can be calculated using UML sequence diagrams. Finally, we have given an overview of our approach. Some of our future works include: (1) comparing our model with other models and showing that our model is optimal. (2) Implementing our model and visualizing results.

8. REFERENCES

[1] Aida Omerovic, Anette Andresen, Havard Grindheim, Per Myrseth, Atle Refsdal, Ketil Stolen, and Jon Olnes 2010, "Idea: a feasibility study in model based prediction of impact of changes on system quality", In Proceedings of the Second international conference on Engineering Secure Software and Systems, pp. 231-240.

[2] Mario Kušek, Saša Desic, and Darko Gvozdanović 2001 "UML Based Object-oriented Development: Experience with Inexperienced Developers", In Proceedings of 6th International Conference on Telecommunications, pp. 55-60.

[3] James M. Bieman, Anneliese A. Andrews, and Helen J. Yang 2003 "Understanding Change-proneness in OO Software through Visualization", In Proceedings of the International Workshop on Program Comprehension

[4] Daniele Romano and Martin Pinzger 2011, "Using Source Code Metrics to Predict Change-Prone Java Interfaces", In Proceedings of 27th IEEE International Conference on Software Maintenance, pp. 303-312

[5] András Pataricza, István Majzik, Gábor Huszerl and György Várnai 2003, "UML-based Design and Formal Analysis of a Safety-Critical Railway Control Software Module", In Proceedings of the Conference on Formal Method for Railway Operations and Control Systems, 2003.

[6] Kathy Dang Nguyen, P.S. Thiagarajan, and Weng-Fai Wong 2007 "A UML-Based Design Framework for Time-Triggered Applications ", In Proceedings of 28th IEEE International Symposium on Real-Time Systems, pp. 39 - 48

[7] Vahid Garousi, Lionel C. Briand and Yvan Labiche, 2006 "Analysis and visualization of behavioral dependencies among distributed objects based on UML models", In Proceedings of the 9th international conference on Model Driven Engineering Languages and Systems, pp. 365-379

[8] Kleanthis C. Thramboulidis 2001 "Using UML for the Development of Distributed Industrial Process Measurement and Control Systems", In Proceedings of IEEE Conference on Control Applications, pp. 1129-1134

[9] A. Güneş Koru, and Hongfang Liu 2007 "Identifying and characterizing change-prone classes in two large-scale open-source products", Journal of Systems and Software, Vol. 80, No. 1, pp. 63-73

[10] Nikolaos Tsantalis, Alexander Chatzigeorgiou, and George Stephanides 2005 "Predicting the Probability of Change in Object-Oriented Systems", IEEE Transactions on Software Engineering, Vol. 31, No. 7, pp. 601-614

[11] M.K. Abdi, H. Lounis, H. Sahraoui 2009 "A probabilistic Approach for Change Impact Prediction in Object-Oriented Systems", In proceedings of 2nd Artificial Intelligence Methods in Software Engineering Workshop, 2009.

[12] Erik Arisholm, Lionel C. Briand, and Audun Føyen 2004 "Dynamic Coupling Measurement for Object-Oriented Software", IEEE Transactions on Software Engineering, Vol. 30, No. 8, pp. 491-506

[13] Daniela Glasberg, Khaled El Emam, Walcelio Melo, and Nazim Madhavji 2000 "Validating Object-Oriented Design Metrics on a Commercial Java Application", National Research Council, September 2000.

[14] Mikael Lindvall 1999 "Measurement of Change: Stable and Change-Prone Constructs in a Commercial C++ System", In Proceedings of IEEE 6th International Software Metrics Symposium, pp. 40-49, 1999.

[15] Erik Arisholm, Dag I.K. Sjøberg 2000 "Towards a framework for empirical assessment of changeability decay", The Journal of Systems and Software, Vol. 53, No.1, pp. 3-14

[16] Ah-Rim Han, Sang-Uk Jeon, Doo-Hwan Bae, and Jang-Eui Hong 2008 "Behavioral Dependency Measurement for Change-Proneness Prediction in UML 2.0 Design Models", In Proceedings of 32nd Annual IEEE International Conference on Computer Software and Applications, pp. 76-83

[17] Ali R. Sharafat and Ladan Tahvildari 2008 "Change Prediction in Object-Oriented Software Systems: A Probabilistic Approach", Journal of Software, Vol. 3, No. 5, pp. 26-40

[18] V.Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and Makan Pourzandi 2009 "Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages", ELSEVIER Electronic notes in Theoretical Computer Science, Vol. 254, pp. 143-160

[19] Mehdi Amoui, Mazeiar Salehie, and Ladan Tahvildari 2009 "Temporal Software Change Prediction Using Neural Networks", International Journal of Software Engineering and Knowledge Engineering, Vol. 19, No. 7, pp. 995–1014

[20] Malan V. Gaikwad, Akhil Khare, and Aparna S. Nakil , 2011 "Finding Proneness of S/W using Class Hierarchy Method", International Journal of Computer Applications, Vol. 22, No. 6, pp. 34-38

[21] Malan V.Gaikwad, Aparna S.Nakil, and Akhil Khare 2011 "Class hierarchy method to find Change-Proneness ", International Journal on Computer Science and Engineering, Vol. 3 No. 1, pp. 21-27

[22] Xiaoyan Zhu, Qinbao Song, and Zhongbin Sun 2013 "Automated Identification of Change-Prone Classes in Open Source Software Projects", Journal of Software, Vol. 8, No. 2, pp. 361-366

[23] Nachiappan Nagappan, Andreas Zeller ,Thomas Zimmermann, Kim Herzig and Brendan Murphy 2010 "Change Bursts as Defect Predictors", In proceedings of IEEE 21st International Symposium on Software Reliability Engineering, pp. 309-318

[24] Emanuel Giger, Martin Pinzger and Harald C. Gall 2012 "Can We Predict Types of Code Changes? An Empirical Analysis", In Proceedings of 9th IEEE Working Conference on Mining Software Repositories, pp. 217-226

[25] Ali R. Sharafat and Ladan Tahvildari 2007 "A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems", In Proceedings of IEEE 11th European Conference on Software Maintenance and Reengineering, pp. 27-38.