A Review of Studies on Change Proneness Prediction in Object Oriented Software

Deepa Godara
Computer Science Engineering
Uttarakhand Technical University
Dehradun, India

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

ABSTRACT
Predicting change prone class in software is a difficult software engineering process. Selection of wrong effort estimation can delay project completion and can incur unnecessary cost also. The aim of this paper is to provide a basis to improve the process of prediction of change prone classes. This paper reports a systematic review of papers published in journals and conference proceedings. The review investigates methodologies for predicting change prone class and fault prone class. The key findings of the review are: (1) behavioural dependency has been widely used for prediction of the change prone class, (2) there is need to develop a framework comprising of more features to accurately predict change prone class. This paper provides an extensive review of studies related to change proneness of software. The main goal and contribution of the review is to support the research on prediction of change prone classes. In addition, we provide software practitioners with useful estimation guidelines (for e.g. classes predicted to be more change prone require more effort).

General Terms
Object Oriented Software, Algorithms, et. al.

Keywords
UML diagrams, change prone class, behavioral dependency.

1. INTRODUCTION
Change-prone classes in software require more attention because they require increased effort, development and maintenance costs. Identifying such classes can enable developers to focus preventive actions such as, peer-reviews, testing, inspections, and restructuring efforts on the classes that are sensitive and change prone. As a result, developers can deliver higher quality products in a timely manner by efficiently utilizing the resources.

Modification may be on account of diverse factors like improvement, modification, perfect upkeep or do away with drawbacks. Several elements of the software may be susceptible to modifications than their counterparts. A proper understanding of the classes which are change-prone is highly advantageous as the change-proneness may be some sign of particular fundamental quality issues [3]. Managing change is one of the pivotal factors in the realm of software engineering. Evolutionary growth has been suggested as a competent method to tackle risks like modern technology and vague or varying needs [15]. If a preservation method has the competence to ascertain the components of the software which are change-prone then explicit corrective steps can be initiated. Therefore, a sound knowledge of the domain which had maximum modification over a certain interval will go a long way in spotting the crucial change-prone classes and interactions, then it is easy for development procedure to concentrate its attention on them [3]. Change-prone classes in software call for meticulous attention as they need endeavor and augment growth and preservation overheads. Recognizing and typifying them enables developers to take remedial measures like peer-reviews, testing, inspections, and streamlining endeavors on the classes with the parallel traits in the coming years. Consequently, developers are capable of exploiting their wherewithal more professionally and dispense superior quality products in an appropriate way [9]. If defective classes are recognized in the initial stages of the growth project's life phase, extenuating remedies can be considered including alert inspections. Forecast patterns by means of plan metrics can be employed to recognize defective classes in advanced stages [13]. The accuracy of the forecast effect decides the accuracy of cost evaluation and quality of project preparation [14]. To obtain behavioral reliance measures between two disseminated objects, we undertake a methodical scrutiny of messages sent between them in a group of sequence diagrams (SDs) For example, when an object sends a synchronous communication to another object and waits for a reply, we say that the former object is behaviorally reliant on the latter [7].

UML is now extensively acknowledged in the software engineering circle as a general notational benchmark. It extends a helping hand to object-oriented plans which on the other hand promote module reprocess. It is competent to furnish numerous outlooks of the system under blueprint [6]. The UML based plan enables us to execute prescribed authentication and corroboration method [5]. The unified modeling language (UML) is a graphical language for visualizing, denoting, building, and recording software-intensive techniques. UML offers a typical method of scripting system plans, covering abstract things, classes written in a definite programming language, database schemes and reusable software components [2]. UML has appeared assuming the role of the software industry's leading language and is, by now, an Object Management Group (OMG) benchmark. It symbolizes a set of finest engineering exercises that have been established as triumphant in the modeling of mega and intricate techniques. OMG is now recommending the UML pattern for global homogeny for information technology [8]. As the application of object-oriented plan and programming grows up in the industry, we see that legacy and polymorphism are being utilized more often to perk up internal reprocess in a system and to assist conservation [12].

We are of the opinion that a lion's share of the software metrics appraise the grade of object-orientation or calculate immobile traits of the plan, which do not appear to be advantageous in finding a key to the perplexing dilemma regarding the existence or otherwise of high-quality in regard to a explicit plan. While attempting to face such an issue, an expert would evaluate the conformance of the plan to well recognized rules of thumb, heuristics, and doctrines [10]. Behavioral Dependency Analysis (BDA) estimates the scope to which the functionality of one system unit is reliant on other units. According to the mine of data used to execute a BDA, we can segregate the BDA methods into three groups such as code-based, execution-trace-based, and model-based.

This paper summarizes empirical results related to change proneness of a class. The primary goal and contribution of the paper is to support the research on change proneness prediction through an extensive review of relevant papers, a brief description of the main results of these papers, and the use of these results to validate change prone classes. Although primarily aimed at other researchers, we believe that most of the paper, in particular the validated features, are useful for software practitioners, as well. The following research questions are addressed here in this paper:

- RQ1: What is the contribution of literature in the field of predicting change prone classes?
- RQ2: What are the methodologies for selecting features for finding change prone class?
- RQ3: What are the techniques used for predicting change prone classes?
- RQ4: What are the change proneness prediction criteria?

Change Proneness Prediction can be formulated as multiple criteria decision making problem. The goal of the CP prediction is:

- To help programmers identify change prone class so that they can focus on such classes
- To help developers deliver higher quality products in a timely manner by efficiently utilizing the resources.
- To help developers focus preventive actions such as, peer-reviews, testing, inspections, and restructuring efforts on the classes that are sensitive and change prone

The remainder of this paper is organized as follows. Section 2 describes the research method applied in this review. The paper is concluded in Section 3.

2. RELATED RESEARCH

2.1 Inclusion Criteria

The main criterion used for including a paper in this review is that paper should describe research in the field finding change prone classes. Only papers that describe: (i) methodology for finding changes, and/or (ii) UML diagrams that helps in finding attributes to predict changes and/or (iii) Change Prediction Technique and/or (iv) System/ tool to assist decision makers in evaluating changes are included in this review. The paper excludes pure discussion or opinion papers. There were examples of papers describing the same study in more than one journal paper. Fortunately, the number of such cases was small and would not lead to important changes in the outcome of the analysis. Therefore such papers are also excluded for that reason.

2.1.1 Contribution of Literature in the Field of Predicting Change Prone Classes

Change-proneness prediction is associated with change impact analysis. The former predicts which classes are likely to change in the future (i.e., change over successive versions), whereas the latter predicts which classes may be impacted by a given change. During the development and maintenance of object-oriented (OO) software, the information on the classes which are more prone to be changed is very useful. Developers and maintainers can make more flexible software by modifying the part of classes which are sensitive to changes. Traditionally, most change-proneness prediction has been studied based on source codes. However, change-proneness prediction in the early phase of software development can provide an easier way for developing stable software by modifying the current design or choosing alternative designs before implementation. A major challenge in software development process is to advance error detection to early phases of the software life cycle. For this purpose, the Verification and Validation (V&V) of UML diagrams play a very important role in detecting flaws at the design phase. It has a distinct importance for software security, where it is crucial to detect security flaws before they can be exploited. . A powerful change prediction tool can improve maintenance and evolution tasks in software projects in terms of cost and time factors. The vast majority of research works have focused on determining "where" the most change-prone entities are, and "how" the change will be propagated through a system. Finding Proneness of software is necessary to identify fault prone and change prone classes at earlier stages of development, so that those classes can be given special attention. Also to improves the quality and reliability of the software. For corrective and adaptive maintenance we require to make changes during the software evolution. As such changes cluster around number of key components in software, it is important to analyze the frequency of changes in individual classes and also to identify and show related changes in multiple classes. Early detection of fault prone and change prone classes can enables the developers and experts to spend their valuable time and resources on these areas of software. Predicting changes in software entities (e.g. source files) that are more likely to change can help in the efficient allocation of the project resources. Identifying change-prone classes can enable developers to pay more attention to classes with similar characteristics in the future and thus test resources and time can be used more effectively. Predicting change prone classes are active key researches is in the field of software engineering. Contribution of literature in the field of predicting change prone classes is given in Table 1.

2.1.2 Methodologies for Selecting Features for Finding Change Prone Class

Authors have used the behavioral dependency measure (BDM) which helps to predict change-proneness in UML 2.0 models. Behavioral Dependency Analysis (BDA) estimates the scope to which the functionality of one system unit is reliant on other units. UML diagrams provide communication between the customer, system analysts and programmers, who write the source code. So, it acts as a mediator or information provider to find out the features. For advance error detection in early stages of software life cycle PROMELA structure is used that provides a precise semantics of most of the newly UML 2.0 introduced combined fragments, allowing the execution of complex interactions. Many authors used matrix and list method, for finding the proneness and dependency of classes. A set of static metrics and change data at class level from an open-source software product, Datacrow can be collected. With this data, Pareto's Law is validated and found

that about 80% of the lines changed are located in only 20% of the classes. Most of the applications of object oriented software used complex inheritance relationship and polymorphism

Table 1. Software studies on software change prediction

No	References	Design of studies	Results
1	[16]	Proposed a systematic method for estimating the behavioral dependency measure (BDM) which enables proper forecast of change-proneness in UML 2.0 brand.	The anticipated measure is estimated on a multi-version medium size open-source project namely JFreeChart. The outcomes clearly exhibited the fact that the BDM is a functional pointer and is competent to be harmonizing to current OO metrics for change-proneness forecast.
2	[17]	Proposed a novel method to forecast modifications in an object-oriented software mechanism	The estimation of change-proneness of components of a software mechanism is an energetic theme in the arena of software engineering. Such evaluation may be profitably used to forecast modification to diverse classes of a system from one version to the next.
3	[18]	offered a formal Verification & Validation method for one of the most admired UML diagrams viz. sequence diagrams to predict inaccuracy in the initial stages of the software life span.	Verification and Validation (V&V) of UML diagrams undertake a very significant function in identifying defects at the planning stage itself. It has a discrete relevance for software safety, where it is highly essential to spot safety faults before they can be subjugated.
4	[19]	invention of innovative Neural Network based Temporal Change Prediction (NNTCP) structure for recognizing the probable location of occurrence of alterations called hot spots	Outcomes established that an awareness of probable time of occurrence of modifications will motivate managers and developers to design their preservation functions with superior proficiency.
5	[20]	a novel a method of employing class hierarchy technique	The new model has successfully spotted change-prone classes and change-proneness of classes and fault-prone classes and fault-proneness of classes. Recognizing the change-prone and inaccuracy prone classes earlier can help concentrating interest on these classes.
6	[21]	Investigated two methods, matrix and list method, employed for estimating the proneness and reliance of classes.	Intelligently 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. In the earlier methods, all needed data was taken manually and from UML diagrams.
7	[22]	gathered a group of static metrics and modification data at class level from an open-source software product, Datacrow	Using this data, Pareto's Law is authenticated. It is also observed that about 80% of the lines transformed are situated in

			only 20% of the classes.
8	[24]	a model for capturing the fine grained Source Code Changes (SCC) and their semantics. Also explored prediction models for whether a source file will be affected by a certain type of SCC. Static source code dependency graph, social network centrality measures and object-oriented metrics are used for predicting details of changes	The results show that Neural Network models can predict categories of SCC types. Proposed model output a list of the potentially change-prone files ranked according to their change-proneness, overall and per change type category.
9	[25]	a novel approach to predict changes in an object oriented software system. The technique uses dependencies generated from the UML diagrams and data obtained from source code of several releases of a software system using reverse engineering. For experimental evaluation a multi-version medium size open source project namely JFlex, the fast scanner generator for Java is used	Result predicts the changes early in the life cycle of the software.
10	[26]	model constructed using network analysis on dependency graph. The dependencies are considered as a low level graph of the entire system.	Evaluation on Windows Server 2003, results that the recall for models built from network measures is by 10% points higher than for models built from complexity metrics. Also experimental studies shows that, network measures could identify 60% of the binaries that the Windows developers considered as critical—twice as many as identified by complexity metrics.
11	[4]	Empirical studies to predict to which extent the existing source code can be used to predict change prone java interfaces. Study of corelation between metrics and number of fine-grained source code changes in interfaces of 10 Java open-source systems	Results show that the external interface cohesion metric exhibits the strongest correlation with the number of changes in source code.
12	[1]	industrial case study predicting impact of architectural design changes on system quality. PREDIQT model	Results shows that PREDIQT model can be used practically in industries with limited resources and efforts
13	[3]	Change prone architecture considering different attributes such as friend functions, coupling, methods over ridden, direct child classes, no of descendants	The method was applied to commercial, embedded real time software to identify and visualize classes and class interactions that are most change prone.

14	[7]	Behavioral dependency analysis of distributed objects using UML sequence diagrams. To visualize dependencies a hierarchical dependency graph is generated.	set of measures are applied to a case study to show its usefulness in predicting behavioral dependencies based on UML models
15	[27]	Proposed method indicates low level quality parts of a software change frequently	Empirical results of the study used to find change prone classes and which class should be tested first.
16	[28]	Derived metrics as potential indicators of the changes prone classes from on release to another	Results show more accurate prediction of class change-proneness is achieved when the evolution-based metrics are combined with product metrics
17	[29]	Presented an empirical study to predict roles that are more change-prone than others and whether there are changes that are more likely to occur to certain roles.	The results are obtained from the source code repositories of three different systems (JHotDraw, Xerces, and Eclipse-JDT) and from 12 design patterns. Results obtained confirm the intuitive behavior about changeability of many roles in design motifs.
18	[31]	Applied statistical meta-analysis techniques to investigate the ability of 62 OO metrics to predict change-proneness	Results from random-effect models reveal that: (1) size metrics exhibit moderate or almost moderate ability in discriminating between change-prone and not change-prone classes; (2) coupling and cohesion metrics generally have a lower predictive ability compared to size metrics; and (3) inheritance metrics have a poor ability to discriminate between change-prone and not change-prone classes
19	[12]	Derived reports on the construction and validation of fault proneness prediction models in the context of an object-oriented, evolving, legacy system.	A cross-validated classification analysis shows that the obtained model has less than 20% of false positives and false negatives, respectively. Results show that when this model is applied to predict faults in a new release, the estimated potential savings in verification effort is about 29%. In contrast, the estimated savings in verification effort drops to 0% when history data is not included.
20	[30]	Presented a novel approach that helps to identify the critical components in the software based on Criticality Analysis	Used several components such as fan in, fan out, information flow, weightage of methods,weakness of methods, ratio of pure inherited methods to calculate criticality.

3. CONCLUSION AND FUTURE RESEARCH

In this paper we present systematic review on prediction of change prone classes in object oriented systems. Related to our research questions we have identified that:

RQ1: What is the contribution of literature in the field of predicting change prone classes?

RQ2: What are the methodologies for selecting features for finding change prone class?

RQ3: What are the techniques used for predicting change prone classes?

RQ4: What are the change proneness prediction criteria?

We have identified some basic problems in the change proneness prediction method. Review shows that UML diagrams also play a vital role in change proneness prediction. Future works for the research community is to (1) focus more on dynamic parameters rather than on static. (2) Consider more parameters in calculation of change proneness prediction (3) focus attention on where the changes will occur rather than when.

4. REFERENCES

[1] Aida Omerovic, Anette Andresen, Havard Grindheim, Per Myrseth, Atle Refsdal, Ketil Stolen, and Jon Olnes, "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, 2010.

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

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

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

[5] András Pataricza, István Majzik, Gábor Huszerl and György Várnai, "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, "A UML-Based Design Framework for Time-Triggered Applications ", In Proceedings of 28th IEEE International Symposium on Real-Time Systems, pp. 39 - 48 , 2007.

[7] Vahid Garousi, Lionel C. Briand and Yvan Labiche, "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, 2006.

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

[9] A. Güneş Koru, and Hongfang Liu, "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, January, 2007.

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

[11] M.K. Abdi, H. Lounis, H. Sahraoui, ―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, "Dynamic Coupling Measurement for Object-Oriented Software", IEEE Transactions on Software Engineering, Vol. 30, No. 8, pp. 491-506, August 2004.

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

[14] Mikael Lindvall, "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, "Towards a framework for empirical assessment of changeability decay", The Journal of Systems and Software, Vol. 53, No.1, pp. 3-14, 2000.

[16] Ah-Rim Han, Sang-Uk Jeon, Doo-Hwan Bae, and Jang-Eui Hong, "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, 2008.

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

[18] V.Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and Makan Pourzandi, "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, 2009.

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

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

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

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

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

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

[25] Ali R. Sharafat and Ladan Tahvildari, "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, 2007.

[26] T Zimmermann, N Nagappan, ―Predicting defects using network analysis on dependency graphs‖, In Proceedings of the 30th international conference on Software engineering,pp 531-540,2008

[27] S Eski, F Buzluca, ―An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes‖ In Proceedings of IEEE Fourth International Conference on Software Testing ,Verification and Validation.,2011, pp-566-571

[28] Mahmoud O. Elish, Mojeeb Al-Rahman Al-Khiaty ―A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software‖ in proceedings of Journal of Software: Evolution and Process Volume 25, Issue 5, pages 407–437, May 2013

[29] Di Penta, M., Cerulo, L.,Guéhéneuc, Y. , Antoniol, G. An empirical study of the relationships between design pattern roles and class change proneness‖ in proceedings of IEEE International Conference on software maintenance,pp 217-226,Sept,2008

[30] D Jeyamala, S Balamurugan, A Jalila ―Fault-prone Components Identification for Real-time Complex systems based on Criticality Analysis‖ In Proceedings of International Journal of Computer Science Issues Vol3,Issue2,pp 17-23,2013

[31] Hongmin Lu, Yuming Zhou, Baowen Xu, Hareton Leung, Lin Chen- The ability of object-oriented metrics to predict change-proneness: a meta-analysis. Empirical Software Engineering June 2012, Volume 17, Issue 3, pp 200-242