International Journal of System Dynamics Applications
Volume 11 • Issue 2 • July-December 2022

Enhancing Behavioral Dependency for Effective Computing in Software

Deepa Bura, Manav Rachna International Institute of Research and Studies, India
Amit Choudhary, Maharaja Surajmal Institute, India

ABSTRACT

Software plays an important role in effective computing and communication of any services. It becomes crucial to identify some critical parts of the software that can lead to enhanced computing and increases efficiency of the software. Dependency plays a significant role in finding relationship amongst classes and predicting change-prone classes. This paper aims to enhance behavioral dependency by defining six types of dependencies amongst classes. These are (1) direct behavioral dependency, (2) indirect behavioral dependency, (3) internal behavioral dependency, (4) external behavioral dependency, (5) indirect internal behavioral dependency, and (6) indirect external behavioral dependency. Evaluating these dependencies gives accurate results for the prediction of change-prone classes. Further, the paper compares the proposed approach with existing methods.

KEYWORDS

Behavioral Dependency, Change-Prone Classes, Communication, Computing, Fault-Prone Classes, Software Engineering, Software Project Management, Software Systems

INTRODUCTION

In the past few years, software industry has grown at a very fast pace. Software systems change constantly with time i.e. every developed software needs to be changed at some point of time in software life cycle. Software change is significant for any organization's progress. As each organization spends a lot of money on their software systems. For maintaining the value of these systems, change is required with changing customer needs.

In any software, there are some parts which are more frequently changed than others. These sensitive parts which are highly prone to changes are known as change prone classes in an object-oriented (OO) software. If such classes are identified early in a software it can help developers to pay more attention on peer review process, testing phase, requirements analysis, maintenance phase and restructuring efforts on particular classes.

UML is a de-facto standard for representing the design of software systems. UML class diagrams depict the dependency among different classes and methods involved in these classes. Thus, it plays a major role in the software development process. When a change in structure or behavior of a class

DOI: 10.4018/IJSDA.20220701.oa1

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

affects the other related class, then there exists a dependency amongst the two classes. So it becomes important to find the change prone classes.

RELATED WORK

Abdeen et al. (2015) predicted improvement, revision, bug fixing and perfective maintenance are also some of the reasons of a software to change. This necessitates software change to be handled properly.

Mathur et al. (2014) analysed that many software projects fail for one or the other reason. One of the major reasons of software failure is incapability to understand the changing requirements and uncontrolled change propagation. Godara et al. (2018) suggested that classes which are prone to changes needs major consideration as these involve more effort and higher amount of maintenance costs and development costs.

Dependency can be defined as degree of association amongst two classes, if change in structure or behavior of one class affects other classes, dependency is said to exist between the classes. Sharafat and Tahvildari (2008) used Unified Modeling Language (UML) diagrams for the evaluation of dependency amongst classes. And the process of reverse engineering was used to find the degree of relationship amongst classes. Jflex software was used for evaluating the results. However, the evaluated results were based on several assumptions.

Lee et al. (2016) worked on co-change, i.e. if one class is changed it affects the other classes also. Research proposed an approach for prediction of co-change volume, using regression line co change was evaluated. Success rate achieved was around 82%. Research focused only on regression line however other factors were ignored. Arisholm et al. (2004) examined change prone classes using dynamic coupling feature. The proposed method was built on relating the amount of modifications in each class with dynamic coupling feature. Godara & Singh (2-15) proposed a new technique to find change in classes using Artificial Bee Colony algorithm.

Accordingly, the proposed model does not fit into the category of change prediction model as effort was not done to associate the anticipated metrics with changes in future versions. The research mainly focussed on finding the relations amongst dynamic coupling and change prone classes. Elish et al. (2014) used the same concept and extended the work of Arisholm et al. (2004) by removing the existing gap of not considering the changes in future versions. Research derived statistical correlation of coupling metrics and change proneness and indicated coupling metrics as a better indicator of change prone classes from one release to another. Software quality is related with software design. High quality software design can benefit in reduction of maintenance and testing costs. Eski et al. (2011) related change prone classes with quality of software. Research indicated software parts which have poor quality tend to change more frequently.

Bura et al. (2017) gave a dynamic measure of predicting change prone classes. Using run time information such as execution time, frequency of methods called, inter-dependency and popularity. Results were validated using OpenClinic and OpenHospital software. Godara and Singh (2014) gave a new hybrid approach for finding change prone classes, in which frequent item set mining algorithm is used to find how many times a method is being called by other methods and how many times a method calls another method. These rules are optimized using Artificial Bee Colony algorithm (ABC) and using decision tree a class is classified as change prone and non- change prone class.

Penta et al. (2008) focussed on prediction of sensitive parts which are more change prone and in addition to this predicted changes which mostly affect some specific classes in a software. The research was based on design patterns which were more affected by change than others. In software evolution, there are design patterns which are more likely to change than others. Considering earlier research, the research was different in the context, as it focussed on certain parts of design patterns rather than focussing on system's entire design pattern.

Godara and Singh (2014) gave a review of different techniques of finding change prone classes and discussed advantages and disadvantages of each method. Further, the paper proposed how

21 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage: www.igi-global.com/article/enhancing-behavioral-dependency-for-effective-computing-in-software/281692

Related Content

A Methodology of Evaluating Service Value based on the Service Field Concept and Its Application to Evaluation of Attractiveness in Sightseeing
Shuang Xuand Michitaka Kosaka (2017). *International Journal of Knowledge and Systems Science (pp. 27-38).*
www.irma-international.org/article/a-methodology-of-evaluating-service-value-based-on-the-service-field-concept-and-its-application-to-evaluation-of-attractiveness-in-sightseeing/169900

Reflexing Interfaces
Franco Orsucci (2008). *Reflexing Interfaces: The Complex Coevolution of Information Technology Ecosystems (pp. 1-20).*
www.irma-international.org/chapter/reflexing-interfaces/28368

Collective Creativity Management in Small and Medium Enterprises: A Case Based Reasoning Approach
Fabio Sartori (2012). *International Journal of Knowledge and Systems Science (pp. 1-23).*
www.irma-international.org/article/collective-creativity-management-small-medium/67084

A Recovery-Oriented Approach for Software Fault Diagnosis in Complex Critical Systems
Gabriella Carrozzaand Roberto Natella (2013). *Innovations and Approaches for Resilient and Adaptive Systems (pp. 29-56).*
www.irma-international.org/chapter/recovery-oriented-approach-software-fault/68942

Dynamically Reconfigurable Hardware for Evolving Bio-Inspired Architectures
Andres Upegui (2010). *Intelligent Systems for Automated Learning and Adaptation: Emerging Trends and Applications (pp. 1-22).*
www.irma-international.org/chapter/dynamically-reconfigurable-hardware-evolving-bio/38448