Towards Improving Object-Oriented Software Maintenance during Change Impact Analysis

Bassey Isong¹ and Obeten Ekabua²
Department of Computer Science, North-West University, Mmabatho, Mafikeng, South Africa
{¹24073008, ²obeten.ekabua}@nwu.ac.za

Abstract - Today, resources are geared towards modifying rather than developing new software systems. Changes are necessary during the system’s lifetime to keep it useful but the major challenge is how these changes are controlled and managed. Software systems are complex with large dependency webs and components that are fault-prone. Modifying components without regard to its dependencies or its fault-proneness may have some unpredicted and potential effects on the quality or increase their risk to fail. Object-oriented software (OOS) systems are not exception. Identifying these components early may reduce system failure risks when implementing changes. Traditional researches on change impact analysis (CIA) of software code change and failure prediction are disjointed. Therefore, the main goal is to propose a change impact analysis framework that incorporates change and failure prediction while enhancing software quality and reducing maintenance time, cost and effort. By way of contribution and extension of existing knowledge, this research will explore and analyze OOS component’s relationship for effective change impact analysis and predicting early, the failure associated with fault-prone components by utilizing OO metrics.

Keywords: Change, impact analysis, Object-oriented, Failure, Metrics, Prediction.

1 Introduction

Software maintenance has been recognized as the most costly and difficult part of software development, accounting for at least 50% of the total software production cost in particular, object-oriented systems [1,2,3]. Software changes are necessary during software maintenance and software might need to be changed to fix defects, to change executing logic, to make the processing more efficient, or to introduce new features and enhancements [5]. However, when changes are made, there will unavoidably have some unpredicted and potential effects on the software and may cause the software to deteriorate. Though software does not deteriorate or change with age, it is believed that most software maintenance involves changes that potentially degrade the software unless it is proactively controlled [4].

Changing OOS in large software systems today is complex requiring a good understanding of the dependencies between software components. This is because a modification to components with little or no regard to dependencies may have some unpredicted and potential effects on the quality of the latter which may increase their risk to fail [6]. Software change impact analysis (CIA) is a technique used to understand and identify the potential effects caused by such changes [2,7]. Given software, the objective is to understand how a proposed change will affect the software components in order to allow more effective prioritization of change requests [1]. An effective CIA can improve the accuracy of required resource estimates, allow more accurate development schedules to be set, and reduce the amount of corrective maintenance by reducing the number of errors introduced as a by-product of the maintenance effort [3].

In the realm of OO maintenance, OO paradigm unlike the procedural paradigm introduce new concepts such as encapsulation, inheritance, polymorphism, and dynamic binding [3,8]. Such features frequently result to more complex relationship between classes and attributes, making it difficult to anticipate and identify the ripple effects of changes. The more a change affects classes, the more its realization cost escalate. In addition, empirical evidences in literature has shown that OO classes are not faults or failures free [9,10]. A software fault is a defect in a software system that may cause an executable product to fail. The intuitive reason is that if a change is implemented on a fault-prone component, software failure will be inevitable. Hence, the early identification of these components will allow mitigating actions to be employed before change can be implemented, if found desirable.

Though several CIA approaches for OOS and software failure prediction exist in the literature [3, 8, 11], the two approaches are disjoint which consequently, can be linked to failures of some OOS after maintenance. It is believed that improving the maintenance of OOS requires CIA approach that is effective at analysis and capturing the complex dependencies among components as well as predicting the early failure of the software, if changes are to be implemented. With this approach, maintenance effort and cost can be reduced while ensuring the quality of the software. In addition, good decisions can be taken before implementing changes. By identifying the potential impact of a modification and the early identification of potential failure, the risk to deal with expensive and unpredictable changes will be reduced. Therefore, the objective of this paper is to...
propose an approach for early failure prediction to support CIA in order to enhance the maintenance of OOS. The approach involves dependencies extraction and analysis, change impact analysis, early failure prediction which will lead to modification decision.

The rest of the paper is organized as follows: the introduction is in section 1, section 2 is a description of the related works, section 3 gives the research goal and approach, and section 4 is work in progress. Accordingly, the research contributions and conclusions are in given section 5 and 6 respectively.

1.1 Background Information

Software changes are inevitable in software development and evolution. Changes occur in every phase of software development like requirements, design, implementation, and maintenance. Thus, systems modification should be taken seriously and the effects of changes must be considered because changes in any phase will affect the behaviour of the delivered software product in another phase [4]. (See Fig. 1)

![Figure 1: Impacts of change on software life-cycle objects](image)

When changes made to software negatively affects the software, it may bring inconsistencies to other parts of the original software and the changed with the affected components may no longer fit for the rest of the software product – software deterioration [4,12]. Deterioration occurs in many ways because changes to software rarely have the small impact they are believed to have [4]. This stems from impact overlapping, impact underestimation to impact overestimation as a result of the complexity and size of current software applications. CIA is a process for controlling changes and avoids software deterioration if properly applied.

In today’s software development world, OO approach is increasingly gaining momentum and widespread use. It is an approach where systems are described in terms of objects. OO approach has the benefits of producing a clean, well-understood design characterized by easier to understand, test, maintain, extend etc [3]. However, the application of the technology does not by itself ensure the quality of the software, guard against developer’s mistakes, nor prevent faults. OO approach introduces new concepts whose features often lead to more complex relationships (i.e. use, invoke, member and inheritance [11]) as shown in Fig. 2. These complex dependencies frequently make it difficult to anticipate and identify the ripple-effects of changes [3,8].

![Figure 2: Dependencies between object oriented program components](image)

For instance, encapsulation promotes an intended functionality to be achieved by invoking several member functions from some classes and changes to a class may affect many classes. Inheritance implies that a class can reuse the data members and member functions of another class. Therefore, new dependencies are created between two classes and changes to a class may affect other classes which are related to it. Polymorphism allows many different implementations of the same specification. All these features make it difficult to define a cost-effective test and maintenance strategy for OOS [3]. With an effective CIA approach, one can determine for some level of granularity (e.g. statements, modules, features), whose components in the software can be affected by changes. In addition, empirical evidence indicates that most OO application components are fault-prone or failure-prone [9,10], though, believed to be found only on few of the system’s components. During the course of maintenance, CIA in particular, if these faulty components are not known before changes are implemented, it could compound the risks and may lead to software failure. Thus, identifying these components prematurely allows mitigating actions, such as validation and verification activities to be focused on the high risk components so as to avoid software failures. Based on this, we intend to evolve a unique failure prediction model that will be incorporated into the CIA process for effective decision making during software changes.

2 Related Works

In this section, we introduce some related current works on CIA. Sharafat and Tahvildari [13] propose a probabilistic approach to predict changes in an OOS system using the dependencies obtained from UML diagrams, as well as source code of several releases of a software system using reverse engineering techniques. Abdi et al. [14] propose the calculation technique of change impact expressions using a meta-model approach to analyze and predict changes impacts in OO systems. Sun et al. [7] propose Object Oriented Class and Member Dependence Graph (OOCMDG) that represents
the program to be analyzed based on static CIA. The objective was on the precision improvement of the impact sets which depends on the change types and the dependence types between the modified entity and other entities. Breech et al. [15] presented coarse-grained impact analysis algorithms that exploit information about how changes can actually propagate due to scoping and parameter passing mechanisms. They present influence mechanisms and describe both static and dynamic impact analysis algorithms that take advantage of these influence mechanisms.

In the same vein, Badri et al. [16] presented a new static technique (CCGImpact) for predictive change impact analysis based on control call graphs (CCG) which captures the control related to components calls and generates the different control flow paths in a program. The generated paths, in a compacted form are used to identify the potential set of components that may be affected by a given change. Oliveira et al. [17] present a hybrid impact analysis technique based on both static and dynamic analysis of OO source code to improve resulting impact estimates in terms of recall. Also, Shao et al. [18] present an approach in which the impact of a source code change can be analyzed by slicing with the variable def-use pairs. Data-flow and program slicing are combined to show data dependencies. Kagdi and Maletic [19] combined the estimated change sets computed from impact analysis techniques with the actual change sets that can be recovered from version histories will result in improved software-change prediction. In the above studies, different CIA approaches on OOS have been reported. All the approaches have been designed for change impact prediction and none employed failure prediction in any way. Therefore, in this research, our approach is unique and is aimed at amalgamating the two approaches in order to effectively calculate the ripple effects and rid or reduce the risks of software failures during and after change implementation.

3 Research Goal

One indispensable property of any software is change and is a key operation for maintenance. These changes are made to realize various change proposals for OOS. With the available change proposal, the maintainer responsible have to analyze and evaluate it in order to predict the impacts in terms of dependencies, and failure-prone components, make a decision on the outcome, and give some modification advice to reduce the risk and cost of the change implementation. For in stance, if a change proposal is known to have significant ripple-effects over the entire system, or undesirable effects or affected classes are fault-prone, the best approach will be either to reject it, or consider an additional change plan, or redesign the system, or accept the change proposal. These activities are carefully carried out before the actual change is implemented and all form parts of the proposed change analysis framework. This research tries to provide an effective and comprehensive solution to the activities related to change analysis in order to improve software maintenance. Therefore, the overall goal of our research is to develop a CIA framework and model for early failure prediction of the impact of changes to OOS to enhance software quality and reduce the time, cost and effort associated with its maintenance.

Figure 3: Proposed CIA framework

Fig.3. presents an overview of our proposed framework. The activities which are the focal points and their corresponding goals are discussed as follows:

1) Dependencies Analysis and Extraction: With the available change proposal and the original OO source code, the first step is to construct a representation for original OOS that is simple and effectively show all the possible dependencies among the components of the original software. The representation is aimed at providing full understanding of how components relates with each other and to facilitate the CIA activities in the next stage.

2) Change Impact Analysis: This step is to perform the actual CIA where the maintainer will have an overview of which parts in original OOS is truly affected by the change proposal, and consequently may bring inconsistence to the software. For effectiveness of this approach, accuracy and precision are top priorities for its evaluation and minimization of the predicted numbers of impact sets.

3) Early Failure Prediction: This is the prediction stage where the change proposal is evaluated from two perspectives - results of the impact analysis and the values of the OO design metrics extracted from the original software for a failure-free change implementation. To predict which of the classes affected by change proposal are fault-prone which in
turn may result in system failure if change is implemented, the extracted OO design metrics and a suitable prediction model will be used and decision made accordingly.

4) Change Implementation: Implementing a change will depend on the results obtained from earlier analysis and evaluation. That is, a change is implemented if the risk is known to be low or after validation and verification activities have been performed on the affected faulty parts. Otherwise, it is rejected if known to have deteriorating effects on the whole system. The essence of the results is to reduce the maintenance time, cost, effort, change consequences and facilitation of regression testing.

4 Research Approach

As stated earlier, our objective is to analyze and predict changes impacts and failure in OOS before change implementation. The approach involves choosing an existing impact model and adjusts it afterward to meet our objective. The work uses both CIA and failure prediction techniques to support and enhance the maintenance of OOS. The approach takes OO components (i.e. field, methods, and classes) and the relationships that exist between them into account as well as the structural properties of the classes. The analysis will be centered on software systems written in Java language which is essential for computing impact of any possible change with our model. In addition, tools (such as Analyst4j) will be utilized to analyze the source code of the system and extract the design measures of every component (class) used for predicting the potential failures.

The approach begins with the construction of an intermediate representation of the original OOS where dependencies between OOS components are extracted and analyzed, and the impact sets are computed. With the impact results, OO design metrics are extracted from the original software and the values are use to predict which components are fault-prone that could result in failure if the change is implemented on such components. With results obtain, decision is then made if a change will be implemented or apply verification activities or reject the proposal if is known to cause huge negative impacts on the entire system.

5 Work in Progress

This section presents how far we have gone with this research. At this point, discussion is based on the investigated dependencies analysis and extraction, CIA techniques and failure prediction approach utilizing the source code change proposal.

5.1 Dependency Analysis and Extraction

Dependency analysis is a critical activity that is performed during CIA. It is an integral part of CIA framework as it assists in understanding how one entity relates to another through effective representation of the original software. Various dependency graphs exist which can be generated by statically extracting the relationships between types (i.e. classes or interfaces) in the source codes. System dependence graph (SDG) [3,20] is one of the commonly used representations for program analysis especially, OO source code. It represents OOS and analyses its elements as well as their relationships at very fine level of granularity [3]. The outcome of the representation is important information about the program elements and relationships between them. Nevertheless, constructing this representation requires much carefulness and good knowledge of OO design because wrong results can lead to over or under estimation of impact sets. Hence, understanding the system dependencies is essential for efficient software change.

Unlike the procedural program, in OO program, emphasis is placed on what the program does to data and their relationships, rather than the program’s structure. In addition, software objects are related to each other by complex dependencies and constraints [11]. To emphasize on this, we use the labeled OO components dependency graph (OOComDG). The OOComDG describes dependencies in OOS while the software system components are viewed as classes, methods, and fields. In our OOComDG, the components are represented as the nodes and the dependences are represented as the edges. The dependences types are the label such as inheritance (H), invocation (V), uses (U), and member (M) on the edges. Once the original software has been represented as OOComDG, it is then transformed two adjacency matrices to ease the correct identification of the starting impact set (SIS). In our initial study, we have constructed a representation of the original system using OOComDG. The dependence between fields and methods, dependence between methods, dependence between classes, and dependence between classes and methods, can be revealed based on the concept analysis. Though some information may be missing in the representation, the representation is in line with various existing activities in the change analysis framework in literature. The initial investigation results show that dependencies between classes, methods and field are well covered, though it is a small sized program. In general, the representation is reasonable and may be applied to large programs.

5.2 Change Impact Analysis

CIA plays an important role in identifying the consequences or ripple-effects of proposed changes. Among
other approaches, static and dynamic analysis are the most commonly used techniques [5,7]. Given the proposed change entities, the object of CIA technique is to find the parts of the software that are truly affected by the change. However, the impact sets produced may be inaccurate due to either underestimate (false-negatives) or overestimate (false-positives) change impact as a result of the problems the maintainer responsible may face in finding the parts affected by the change.

To guard against these inaccuracies and reduce the predicted impact sets, our research employed static CIA technique using concept of OO impact method to compute the potential impact set from the proposed changed. In view of this, we will introduce the impact range concept to obtain the impacted entities in a given change category based on movement along the program’s OOComDG from the SIS of the changed entity obtained from the adjacency matrix. In OOS, different change types often have different impact methods. For instance, some changes made to programs do not affect other entities in programs regardless of some dependencies while some other changes may potentially impact other entities in programs [11]. The impact method of a change is based on the code change types of modified components and dependencies between them (i.e. the modified and other components). In our approach, the SIS is computed using the adjacency matrices, while the potential impact set (PIS) is computed based on the SIS and impact range. Two types of adjacency matrices are introduced here: intra and inter-class member relation matrices. At this point, we have not validated our techniques on the real-world program to see its effectiveness. However, we are confident the technique will produce fewer impact sets with a reduction in false-positives and false-negatives.

5.3 Early Failure Predictions

The early failure prediction is a stage where we evaluate the probability of failure occurrence if a change proposal is implemented. It is based on two aspects: results of the impact analysis and the values of the OO design metrics extracted from the original software which is then mapped onto the affected classes. It is true that when changes are made to software, they will inevitably have some unpredictable and undesirable ripple-effect on other parts of the software [21]. In the same way, the degree of the ripple-effect is proportional to the complexities of the structural properties of the software product which in turn affects the cognitive complexity of the maintainer. Cognitive complexity is known to constitute the mental burden of the maintainer who has to deal with the component [10]. Thus, high cognitive complexity of a system leads to components exhibiting undesirable external qualities, such as increased fault-proneness and reduced maintainability (see Fig. 4).

Figure 4: Effect of product’s structural properties on maintenance

Several empirical evidences have shown that OOS applications are not fault-free, though found only on few of the system’s components [9]. In addition, our perception in this research is that when changes are implemented on components that are fault-prone, they will complicate the situation and inevitably result in failure. Therefore, predicting the failure early before implementing change proposal can help maintainer responsible to answer whether a change proposal is accepted or to determine which change schedule is more suitable to employ (i.e. to focus verification actions on the high risk, failure-prone components) or to decide on rejecting the change proposal.

In the sphere of OOS, the construction of prediction models using OO design metrics is one approach aimed at discovering faulty classes early in development and maintenance. Such models usually uses historical data, design metrics which can used for identifying potentially faulty classes in future applications or releases [10]. With the result of such investigation, an organization can take actions prematurely aimed at alleviating the situation and consequently avoid costly rework. Several numbers of OO design metrics have been constructed such as the Chidamber and Kemerer [22], Li and Henry [23], Abreu and Carapuca [24], Briand et al. [25], etc. in literature. In addition, there are several empirical studies that validated and revalidated the relationship between the different OO design metrics and fault-proneness and their effect on cognitive complexity as well as the prediction models that utilize them in the literature [26,27,28,29].

In this research, we are going to employ the existing OO design metrics, particularly, the Chidamber and Kemerer [22]. Emphasis will be on extracting the design metrics that are positively associated with the fault-proneness of classes. However, the question is, “which metrics are suitable for OO failure prediction?” Though several prediction models associated with fault-proneness exist, the approach of this research will be unique and two-dimensional. This means that we will consider failure prediction based on the values of extracted measures for both pre-release and post-release OOS. For prediction based on design measures, the intuition is that higher values of these metrics represent structural properties that increase the probability that a class will have a fault that causes a field failure. At this point, we are still at
the stage of identifying which OO design metrics are significantly associated with fault or failure-proneness.

6 Research Contributions

Understanding changes during CIA is essential for understanding the evolution of a software system. With our proposed approach, given a change proposal, the task is i) obtain the information about the dependencies in original system, ii) compute the potential ripple-effects induced by the change proposal based on the code change type and impact and dependency types, iii) perform the early failure prediction based on the design measures extracted from the original system to identify fault-prone components that could cause failure if change proposal is implemented, and iv) make modification based on the results. Consequently, the expected contributions of the research are as follows:

- Support maintainer in performing static CIA on OOS through:
  - A representation that is simple enough and reveal all the dependencies between the elements in the original software
  - Capture impact sets that are accurate, not large enough or difficult for practical use and with fewer false-negatives and false-positives to predict the ripple effects induced by the change proposal.

- To evolve software metrics (i.e. predictors) that is based on the structural properties of the product and which can accurately predict failure early enough that is assumed to occur when certain changes are made.

- CIA framework that support various change analysis activities by incorporates impact prediction and failure prediction in order to identify and reduce the cost and risks associated with change implementation.

In all, by identifying the potential impacts and failures before change implementation during maintenance, the risks associated with embarking on a costly change can be reduced drastically.

7 Conclusions

In this paper, we have proposed an approach to support the maintenance of OOS system during CIA through early failure prediction. The approach starts with dependencies analysis and extraction of the original software, impact analysis based on adjacency matrix analysis and their impact expression and early failure prediction based on extracted design metrics and historical data. Although, the research is still at its preliminary stage, we however conclude that, by identifying potential impacts and failures before committing a change, the risks associated with embarking on a costly change will be drastically reduced. This is because the cost of unforeseen problems generally increases when there are discovered lately. Furthermore, it will help management to choose between alternative changes when undesirable effects are known. The work is still in progress with emphasis on the CIA and failure prediction phases.

8 References


