Computer and Information Science; Vol. 6, No. 3; 2013
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

An Assessment of Changeability of Open Source Software

Yirsaw Ayalew[1] & Kagiso Mguni[1]

[1] Department of Computer Science, University of Botswana, Botswana

Correspondence: Yirsaw Ayalew, Department of Computer Science, University of Botswana, Botswana Private Bag 0704, Gaborone, Botswana. Tel: 267-355-2869. E-mail: ayalew@mopipi.ub.bw

Received: April 18, 2013 Accepted: May 9, 2013 Online Published: May 23, 2013
doi:10.5539/cis.v6n3p68 URL: http://dx.doi.org/10.5539/cis.v6n3p68

Abstract

Among the maintainability sub-characteristics, changeability plays a critical role in analyzing the maintainability of software. Changeability is highly influenced by the dependencies that exist between the components of a system. Therefore, we need to have measurement mechanisms that can take into account the dependencies between components so as to determine the ease which modifications can be made to given software. This paper attempts to investigate the usefulness of three coupling metrics (CBO, Ce, and Ca) and one size/complexity metric (WMC) as predictors of changeability. The assessment is based on a case study of open source software known as OpenBravoPOS which is commonly used in the retail business and developed in Java. The results show that some of the coupling metrics can be used as good predictors of changeability.

Keywords: changeability, coupling metrics, maintenance, OpenBravoPOS

1. Introduction

Software maintenance is the most expensive activity that consumes about 50–70 percent of development cost (Jalote, 2005). For this reason, people have attempted to find ways to minimize maintenance costs by introducing better development methodologies that can minimize the effects of change, simplify understanding of programs, facilitate early detection of faults, etc. Before a maintenance task is carried out, it is crucial to assess the maintainability of the software under consideration. According to IEEE (1990), maintainability can be defined as *"The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment"*.

In order to measure maintainability, we need to employ appropriate metrics. There are various maintainability metrics of which some are based on design properties of a system while others are based on code level properties of a system. In their study, (Riaz, Mendes, & Tempero, 2009) found that metrics based on size, complexity, and coupling are the most successful maintainability metrics. Moreover, among the successful predictors, most are gathered at the source code level. However, they have also discovered that there are no studies that attempted to explore the treatment of maintainability metrics and their applicability on different maintainability sub-characteristics and maintenance types. In another study, Yuming and Baowen (2008) conducted an experimental study on the relationship between design metrics and maintainability of open source software. The results of their study indicate that size and complexity metrics are strongly related to the maintainability of open source software. Dagpinar and Jahnke (2003) found that size and direct coupling metrics are significant predictors for measuring maintainability of classes. In a major critique to studies on maintainability (Broy, Deissenboeck, & Pizka, 2006), argued that "there is no common understanding of what maintainability actually is, how it can be achieved, measured, or assessed". To address this issue, they proposed a two-dimensional quality model that treats maintenance activities and system properties separately.

With regard to open source software, empirical studies show that open source software is more maintainable than proprietary software (Samoladas, Stamelos, Angelis, & Oikonomou, 2004). These studies used design metrics for comparing the maintainability of open source software and proprietary software. However, (Schach, Jin, Wright, Heller, & Offutt, 2002) demonstrated that after a long period of maintenance, even an open source software may become difficult to maintain. In other words, maintainability deterioration over time is expected for both open source software and proprietary software system.

Among the maintainability characteristics, changeability plays a critical role in analyzing the maintainability of software. Changeability refers to how easy it is to perform a specified modification (ISO/IEC9126-1, 2001).

Many studies assessed changeability based on change impact analysis (CIA). CIA is an approach used to identify the potential ripple effects caused by changes made to software (Bohner & Arnold, 1996). CIA techniques focus on determining the effects of the proposed changes on other parts of the software. Sun, Li, and Zhang (2012) proposed an extension of the CIA technique by introducing an impactness metric which takes individual change proposals into account and determines the degree to which the software can absorb the change proposal. For this purpose, they introduced the formal concept analysis which is based on a lattice diagram constructed for a given change proposal. Other studies focus on the development of changeability assessment models. For example, Fluri (2007) proposed a changeability assessment model for source code entities. They provided taxonomy of source code changes and used this taxonomy with a set of changeability criteria to assess the changeability of an evolving system. Along this line, there are a number of techniques proposed for change impact analysis. Li, Sun, Leung, and Zhang, (2012) conducted a comprehensive survey study to determine the application of existing code-based CIA techniques in software maintenance. Their results show how CIA techniques are increasingly becoming crucial for the assessment of software evolution. However, there are not many studies that investigated metrics that can effectively be used to assess changeability of open source software systems.

In this study, we are interested in to assess the relationship between metrics and changeability of open source software. Specifically, we ***would like to investigate design metrics that can be used as good indicator of software changeability.*** To do so, we assessed the changeability of the evolving open sources software system – OpenBravoPOS by analyzing its change history. The focus of this paper is on analyzing fine-grained source code changes.

The rest of the paper is organized as follows: Section 2 presents related work on changeability assessment. A discussion on how we conducted the study is provided in Section 3. The results of the study and issues emanating from the study are discussed in Section 4. Finally, Section 5 presents the main points of the study and future work.

2. Related Work

In the literature, there are different approaches used for the assessment of changeability. Most assessments are based on change impact analysis (using changeability model) and some are based on design metrics. A discussion of related work from both approaches is provided below.

2.1 Change Impact-Based Approaches

Grover, Kumar, and Kumar (2008) proposed a measurement approach for evaluating the changeability of an aspect oriented (AO) system by assessing the changes made at the code level. This measurement approach specifies the minimum changes required in order to properly evaluate changeability. For example, for a component with 3 operations, a minimum of 12 changes are required to evaluate Average Change Impact (ACI). For each component, they computed the ACI and the number of dependencies (measured with the help of adjacency matrix). They found that as the dependency among components increases, the impact of change increases which indicates low changeability. They concluded that a highly dependent AO system will absorb low changeability.

Chaumun, Kabaili, Keller, and Lustman (1999) proposed a change impact model that aims to assess the impact of change when a maintenance task is carried out on a system. The changes proposed in the model are carried out at the field, method or class level. Their impact model is based on the types of changes, their links, and the impact. Using this model, they computed the impact of changes made to classes of an object-oriented system implemented in C++. They considered only the method signature change in their experiment. The result showed that the software system used for the experiment was able to easily absorb that kind of change and that well chosen OO design metrics (e.g., WMC) can be used as indicators of changeability.

Sun et al. (2012) proposed FCA-based Impact Analysis to assess the changeability of a system when presented with a change proposal. By using formal concept analysis, they demonstrated how to perform change impact analysis which estimates the ripple effects of the change proposal. With the use of the impactness metric it was possible to measure the system's changeability to absorb change proposals. Their case study results show that their changeability assessment approach can provide more effective changeability information as compared to those approaches that rely only on the impact results.

Arisholm (2006) carried out a study to assess the impact of structural properties of software on changeability. In their study, they assessed the structural attributes and the change profile metrics. The structural metrics considered were system size (LOC as in CK Metrics), Class Count and Class Size, Method Count (WMC as in CK Metrics) and coupling metrics. The change profile metrics were calculated as the number of lines of code added and deleted from snapshot j to j-1. It was found that the change profile measures are better to measure maintenance effort. They also extended the definition of changeability by adding that a change should avoid degrading software from a satisfactory state to an unsatisfactory state. In other words, being capable of change should result in minimum degradation of the software quality.

2.2 Design Metrics-Based Approaches

Chaumun et al. (2000) used the CK Metrics for evaluating the changeability of OO software. They argued that while the CK metrics have been used as good predictors of maintainability, the metrics were not assessed in relation to changeability. The metrics considered in their study were WMC, DIT, NOC, CBO and RFC. In their study they tried to determine a correlation between the number of classes impacted by a given change and the CK metrics. They redefined the CBO metric and introduced the CBO_NA (CBO No Ancestors) metric to define a coupling that does not include coupling to the ancestors. In their study, the changeability of three systems was assessed: XForms which is a small sized graphical user interface application; ET++ which is a medium sized application framework, and System-B which is a large sized system. The results of their study show that there is a strong correlation between changeability and coupling to methods and fields. In particular, their CBO_IUB (CBO Is Used By) metric was the most highly correlated with changeability. The CBO_IUB represents the part of CBO that consists of the classes using the target class. In addition, they found out that DIT is not correlated with changeability.

Ingram and Riddle (2011) proposed a method for predicting change proneness in software by demonstrating a correlation between software size/complexity and change proneness. Their study was focused on one package (e-Science pilot project developed in Java) which had two iterations in two years. In their study six metrics were collected: LOC, DIT, WMC, CBO and McCabe's cyclomatic complexity. Change proneness was measured as the number of files changed for each revision. They found out that classes with the highest CBO were the most likeliest to change.

Wilkie and Kitchenham (2000) assessed the usefulness of the CBO metric in predicting the classes that are likely to be affected by a given change. One of their research questions was whether classes with a higher CBO values are more likely to be affected by a change originating in another class. Their case study was a C++ commercial multimedia conferencing system which has 25, 000 lines of code. In their study, the CBO, WMC and NPM metrics were measured. The features analysed for the study included addition of new features and bug fixes. They found that the CBO metric is useful in identifying the most change prone classes but not the classes likely to experience ripple effect changes. As a result, they recommended for a more comprehensive measure which can predict change ripple effects.

In this study, we focus on changeability assessment based on design metrics. Most studies that used metrics to assess changeability used the CK metrics. Among the CK metrics, CBO and WMC are found to be useful in assessing changeability. Some studies (e.g., Chaumun, Kabaili, Keller, Lustman, & Saint-Denis, 2000), used modified versions of the CBO metric and found that this modified version can also be indicator of changeability.

We believe that changeability of a system is better assessed in terms of coupling metrics as coupling metrics measure the dependencies among various objects of a system. Coupling is a measure of the degree to which two classes in an object-oriented system depend on one another. These dependencies can serve to assess the impact of change as changes can propagate from one class to other classes through the dependencies. A class that is coupled to other classes is sensitive to changes and as a result it becomes more difficult to maintain and gets more error-prone. Additionally it is harder to test a heavily coupled class in isolation and it is harder to understand such a class. Changeability of a class (and hence a system) depends on the number of changes required in the class and the impacts of these changes on other classes. For example, a change prone class will lead to more changes applied to it where the changes are a result of changes necessary in its functionality or bug fixes or necessitated by changes on other classes upon which it depends. Therefore, those metrics that capture the dependencies among classes will have a better likelihood of indicating changeability.

Therefore, our case study investigates the suitability of the coupling metrics Ca (afferent coupling), Ce (efferent coupling), CBO, and WMC to the assessment of changeability of evolving open source software implemented in Java. We are aware that there are many other structural coupling metrics as documented in Poshyvanyk and Marcus (2006). However, for this study, we focus on CBO, Ca, and Ce together with WMC. The coupling metrics Ca and Ce have not been investigated adequately for their capability in assessing the changeability of a system despite their presence in many tools that are used for code analysis (e.g., JDepend). The Ca and Ce metrics have been used to assess modularity of a system which is a characteristic of maintainability in that modular systems are easy to maintain. Therefore, metrics applicable for assessing modularity are expected to have the ability of predicting changeability of a system.

3. Study Setup

The main aim of this research is to determine the metrics that can be used as good indicators of software changeability. To do so, we collected historical maintenance data from an open source software system known as OpenBravoPOS.

3.1 Selection of OpenBravoPOS

The main reason for the selection of OpenBravoPOS system is because we have experience in the development of the system. OpenBravoPOS is a Java-based application that implements functionalities for the retail business (OpenBravoPOS, 2008). It is an open-source system built with several components which include Apache poi which is used for reading and writing Microsoft Office files, iText which is used for reading and writing PDF files, Commons-Logging which is a component that provides logging features, and jasperreports which is used for the generation of reports.

To get reliable assessment of changeability, we wanted to use existing historical data about maintenance records of real-world example systems. This helps us to avoid bias as the maintenance tasks had already been carried out by people different from the researchers who carry out the changeability assessment. This also helps to achieve a more objective assessment as we are not trying to predict future maintenance effort.

3.2 Data Collection

Version 2.20 of OpenbravoPOS was forked from version 2.10 after over 70 maintenance tasks. The maintenance log files were accessed from the **SVN** repository of the OpenbravoPOS project. These maintenance tasks fall into the following categories:

- Bug fixes – Bug fixes are maintenance tasks that are carried to resolve bugs that exist in the software. These bugs were tracked using **Bugzilla** bug tracking system for version 2.20 of OpenbravoPOS. Version 2.30 and upwards use the **Mantis** bug tracking system and **git** for source code versioning.
- Addition of new features – This represents the addition of new features to the existing feature set of OpenbravoPOS.
- Refactoring – This represents the changes to the source code of OpenbravoPOS that leave the functional state unchanged. An example of a refactoring is a removal of dead code.

Each revision of the OpenbravoPOS represented a separate maintenance task and the impact of change in response to this task is the number of components affected by the change. In the **SVN** repository, the log messages were used to classify the maintenance tasks into either a bug fix, feature addition or refactoring. The bug fixes log messages start with the word "**Fix**" and in some cases they also included bug ids. The feature additions had the word "**add**" or variants of the word add. Refactoring tasks had the word "**refactoring**" or "**code clean-up**".

The different maintenance tasks were mapped to the corresponding program components in the source code using the Feature Exploration and Analysis Tool (FEAT) (Robillard, 2005). FEAT has been used for mapping the components related to a maintenance tasks. The maintenance tasks were identified by the revision ID. In the SVN repository the changes made can be viewed at the code level hence they can be easily identified and then manually assigned to the different program units.

The commit messages in the SVN repository included a description of the changes and also showed a list of modified files. The metrics collected in this study are the OO metrics, while the changes made to the OpenbravoPOS are not limited to only Java files. Some of the changes are linked to other files such as .properties, .sh, .bat and also .sql files. Maintenance tasks carried out on non-java programs were ignored in this study as our metrics suite only focused on OO metrics. Moreover, we relied only on the SVN commits and not the issue tracking system as the focus of the study is on the maintenance tasks that had already been carried out. Bugs that are not fixed means that they cannot be linked to the different program elements hence they are ignored in this study. We followed the same definition as Eaddy et al. (2008) in linking the program elements with a maintenance tasks, i.e., a program element is relevant to a maintenance task if it was modified or added to carry out the maintenance task.

An example of the log message from the SVN repository is shown below:

```
Revision: 178
Author: adrianromero
Date: 6:14:10 PM, Wednesday, May 28, 2008
Message:
Fixes issue 1976384 AT210: The customers list does not work in Oracle
----
Modified : /trunk/src-pos/com/openbravo/pos/customers/CustomersPanel.java
Modified : /trunk/src-pos/com/openbravo/pos/customers/CustomersPayment.java
Modified : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomers.java
Added : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomersCreate.java
Added : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomersHSQLDB.java
Added : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomersMySQL.java
Added : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomersOracle.java
Added : /trunk/src-pos/com/openbravo/pos/customers/DataLogicCustomersPostgreSQL.java
Modified : /trunk/src-pos/com/openbravo/pos/payment/JPaymentSelect.java
Modified : /trunk/src-pos/com/openbravo/pos/reports/JParamsCustomer.java
Modified : /trunk/src-pos/com/openbravo/pos/sales/JPanelTicket.java
Modified : /trunk/src-pos/com/openbravo/pos/sales/restaurant/JTicketsBagRestaurantRes.java
```

The above log file from OpenbravoPOS shows that some of the code was modified and others were added as a result of fixing bug 1976384. For example, the ***Customerspanel.java*** file was modified and the ***DataLogicCustomers.java*** was added. In addition to showing the files that were modified the SVN client also allows us to see the changes at the code level. This helped us to map the program elements to the maintenance tasks.

4. Data Analysis and Results

4.1 Maintenance Tasks

OpenBravoPOS has 119,269 lines of code (LOC). Among this, Java code is 62,417 LOC. The rest of the code has been written in a combination of other language such as XML, HTML, CSS, SQL, JavaScript, etc. The LOC reported here excludes comments. This study analyzes only the Java code.

Table 1 provides a summary of the maintenance tasks and the impacts on the system. It shows that most of these maintenance tasks were bug fixes.

Table 1. Maintenance tasks and their impact

Maintenance Type	No. of maintenance tasks/changes	Components affected	Average impact of change
Bug fixes	34	117	3.44
New features	20	433	21.65
Refactoring	8	21	2.63
Total	62		

A component in Table 1 refers to fields, methods, or classes. According to Chaumun et al. (2000), there are 12 types of changes that can affect a component. These 12 types of changes are as follows:

- Variable or Field - Addition, deletion, type change and scope change
- Method – Addition, deletion, implementation change, signature change and scope change

- Class – Addition, deletion and structure change

In addition to the above 12 changes, we identified 2 more types of changes that could be useful during the assessment of changeability. These two changes are the renaming of a field and a change of a field. The change of a field includes such changes as changing the static declaration of a field or changing the initial value of a field. The following table provides the list of change types and their frequencies during the maintenance of OpenBravoPOS.

Table 2. Different change types and their impact

Type of change	Frequency
Field deletion	21
Field added	78
Field type change	2
Field renamed	3
Field change	5
Field scope	11
Method implementation change	304
Method signature	6
Method deleted	43
Method added	91
Class deleted	4
Class structure change	3
Class added	20

Table 2 shows that method implementation change was the most common maintenance task followed by the addition of methods and fields. Overall methods were the most affected by the maintenance tasks as expected.

4.2 Impact of Change

Different changes in the source code have different impact on the system. Some changes are localized that they do not affect other components (field, method, class) of the system whereas others affect a number of components. This depends mainly on the relationship between a component where the change originates (sometimes called base component) and the other components which depend on the base component. For example, in our analysis, we found that some changes affect only 1 component while another affected up to 110 components. The following table shows our analysis of the impact of change from the historical maintenance data. We distinguish between the impact of change as a result of addition of new features and the impact of change as a result of fixing faults. The table below shows a comparison between the impacts brought about by fixing faults as compared to that brought about by addition of new features.

Table 3. Comparison of Impact of change due to addition of new features and due to bug fixes

	Min # components	Max # components	Average # components	Min # classes	Max # classes	Average # classes
Impact of Changes due to addition of features	1	110	22	1	28	7
Impact of Changes due to fixing of Faults	1	24	3	1	12	2

Among the changes due to addition of new features, the maximum number of components affected was 110 whereas the average was 22. On the other hand, the maximum number of components affected as a result of bug fix was 24 whereas the average was 3.

According to Chaumun's model the impact of change is dependent on the links and the types of change. For example, in revision 215, changing fields from private to protected resulted in the corresponding fields removed from the sub-class. In addition, a new field called ***search key*** was added to the ***CustomerInfo*** class and the constructor was modified. Following the modification of the signature of the constructor, all calls to this constructor were modified.

4.3 Coupling Metrics

As indicated earlier in Section 2 (**related work**), in this study we focus on investigating the capability of coupling metrics to serve as indicators of changeability. Many studies have already investigated the CBO and WMC metrics for this purpose. However, the coupling metrics Ca and Ce have not been studied adequately as predictors of changeability. Therefore, in this study, in addition to the well studied metrics CBO and WMC, we investigate Ca and Ce for their capability in predicting changeability.

We used the CKJM extended (Chidamber and Kemerer Java Metrics) tool for calculating the metrics. The definitions of the different metrics are given below:

Table 4. Definition of metrics

Metric	Definition
WMC (weighted method per class)	represents the sum of the complexities of its methods
CBO (coupling between objects)	represents the number of classes coupled to a given class. This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions
CA (Afferent Couplings**)**	a measure of how many other classes use the specific class
CE (Efferent Couplings**)**	a measure of how many other classes are used by the specific class

Table 5 shows the different classes that had changes and their corresponding values of the different metrics.

Table 5. Coupling and complexity metrics

Class Name	**CBO**	**CA**	**CE**	**WMC**	**# maintenance tasks**
com.openbravo.pos.sales.JPanelTicket	77	19	72	69	8
com.openbravo.pos.forms.DataLogicSales	42	14	38	42	4
com.openbravo.pos.sales.JTicketsBagTicket	31	7	31	21	3
com.openbravo.pos.ticket.TicketLineInfo	10	1	9	42	3
com.openbravo.possync.DataLogicIntegration	28	13	20	13	3
com.openbravo.data.loader.Session	12	12	0	11	2
com.openbravo.pos.forms.BeanFactoryData	7	2	5	4	2
com.openbravo.pos.panels.JPanelCloseMoney	25	2	25	12	2
com.openbravo.possync.DataLogicIntegration$2	10	3	10	2	2
com.openbravo.pos.inventory.TaxEditor	10	1	9	9	1
com.openbravo.pos.panels.PaymentsModel	15	3	14	30	1
com.openbravo.pos.payment.JPaymentDebt	15	2	14	7	1
com.openbravo.possync.OrdersSyncCreate	6	0	6	2	1
com.openbravo.pos.panels.PaymentsEditor	11	1	10	9	1
com.openbravo.pos.panels.PaymentsModel$2	3	1	3	5	1
com.openbravo.pos.payment.JPaymentSelect	30	18	17	30	1
com.openbravo.pos.sales.JProductLineEdit	18	7	17	19	1

Class Name	**CBO**	**CA**	**CE**	**WMC**	**# maintenance tasks**
com.openbravo.possync.ProductsSyncCreate	7	0	7	2	1
com.openbravo.pos.forms.BeanFactoryScript	9	1	8	3	1
com.openbravo.pos.reports.JParamsCustomer	13	2	13	10	1
com.openbravo.pos.customers.CustomersPanel	16	0	16	10	1
com.openbravo.pos.customers.CustomerInfoExt	6	3	3	41	1
com.openbravo.pos.config.JPanelConfigGeneral	18	9	17	22	1
com.openbravo.pos.customers.CustomersPayment	37	5	37	23	1
com.openbravo.possync.DataLogicIntegration$4	12	3	12	2	1
com.openbravo.pos.customers.DataLogicCustomers	31	12	24	11	1
com.openbravo.possync.DataLogicIntegrationMySQL	7	1	7	3	1
com.openbravo.possync.DataLogicIntegrationCreate	1	0	1	1	1
com.openbravo.possync.DataLogicIntegrationHSQLDB	7	1	7	3	1
com.openbravo.possync.DataLogicIntegrationOracle	7	1	7	3	1
com.openbravo.pos.customers.DataLogicCustomersMySQL	1	0	1	1	1
com.openbravo.pos.payment.JPaymentCashPos$AddAmount	5	1	4	2	1
com.openbravo.pos.customers.DataLogicCustomersCreate	1	0	1	1	1
com.openbravo.pos.customers.DataLogicCustomersHSQLDB	1	0	1	1	1
com.openbravo.pos.customers.DataLogicCustomersOracle	11	1	11	2	1
com.openbravo.possync.DataLogicIntegrationPostgreSQL	7	1	7	3	1
com.openbravo.pos.forms.StartPOS$1	8	1	8	2	1
com.openbravo.pos.catalog.JCatalog	22	8	22	28	1
com.openbravo.pos.admin.PeopleView	15	4	14	15	1
com.openbravo.pos.util.RoundUtils	1	0	1	4	1
com.openbravo.pos.forms.AppConfig	2	1	1	14	1
com.openbravo.data.gui.MessageInf	6	5	2	11	1
com.openbravo.pos.forms.JRootApp	36	10	31	36	1
com.openbravo.pos.customers.DataLogicCustomersPostgreSQL	1	0	1	1	1
com.openbravo.beans.JNumberKeys	2	1	2	13	1
com.openbravo.pos.forms.AppUser	12	9	5	18	1

In order to determine the suitability of the metrics for the assessment of changeability of the system, we computed the Pearson correlation coefficients between the metrics used in the study and the maintenance tasks. Table 6 shows the correlation coefficients.

Table 6. Pearson correlation coefficients

Metric	Pearson correlation coefficient (r)
CBO	0.741996
CA	0.56808
CE	0.736076
WMC	0.660035

The Pearson correlation coefficients show moderate to strong correlations between the number of maintenance tasks and the metrics. Among the metrics, the CBO shows a strong positive correlation followed by the Ce followed by WMC and then the Ca. Commonly, a value of $r > 0.7$ represents strong positive correlation and a value of $0.3 < r < 0.7$ represents moderate correlation. Therefore, CBO and Ce show strong correlations while WMC and Ca show moderate correlation. In some cases, a value of $r > 0.5$ is considered strong. Therefore, the selected metrics are strongly correlated to maintenance tasks. That means as more maintenance tasks are applied to a class, the values of the CBO, Ce, Ca, and WMC increase. The increase in the values of the metrics indicates that more effort is required to carry out the changes. Therefore, as the values of these metrics increase, the changeability of the classes and hence the system decreases. Hence, these metrics are inversely proportional to changeability.

5. Disscussion

5.1 Metrics and Changes

From the Pearson correlation coefficients, all the coupling metrics and the WMC have a strong positive correlation with maintenance tasks. This shows that the metrics considered in this study (i.e., CBO, WMC, Ca, and Ce) are good indicators of changeability. As the values of these metrics increase, the changeability of the system decreases. As shown by Dubey and Rana (2011), maintainability is inversely proportional to the metrics CBO, WMC, RFC, LCOM, DIT, and NOC. Therefore, from our correlation result, we can see that CBO, WMC, Ca, and Ce are inversely proportional to changeability.

CBO has the highest correlation coefficient among the metrics used in this study. As the number of changes increases, the CBO values increases. This is because any maintenance action on a class will propagate the action to all the classes depending on that class. Thus all the classes depending upon a class have to undergo a change every time a class changes. Thus, as the number of classes accessing a particular class increases, the CBO value increase and hence more maintenance effort is required. This degrades the changeability of a class.

Another perspective to look at is the percentage of classes with more than the recommended CBO values. According to Sahraoui, Godin and Miceli (2000), the maximum CBO value should be 14 as higher values have negative impacts on several quality aspects of a class, which includes the maintainability, stability and understandability. But from the Table in appendix A, we can observe that more than a third of the classes have CBO value of more than 14. This indicates the degree of changeability of the system.

Among the metrics, Ce has the second highest correlation coefficient. A high value for Ce indicates that a class depends on many other classes which could be difficult to deal with during changes. Classes that have a high efferent coupling may be inversely related to changeability, since they are using other classes of the system which might need to be changed as well. Therefore, changeability decreases with increasing Ce. We can also see that other sub-characteristics of maintainability (e.g., analyzability, adaptability, etc.) decrease with increasing Ce and hence high Ce affects maintainability of a system. This result is also in line with other studies such as Arisholm (2002) and (Emam, Melo, & Machado, 2001) which found that efferent coupling being stronger at predicting class quality when compared to afferent coupling.

WMC represents a weighted sum of methods implemented within a class which in a way also indicates the complexity of a class. A high value in WMC indicates that classes are more complex and hence require a significant effort to change. Therefore, as WMC increases the changeability decreases. Changing a class requires prior understanding, which, in turn, is more complicated for large and complex systems. Since WMC also allows an assessment of size or complexity, changeability declines with increasing WMC.

The inward coupling of classes is represented by the Ca values. This indicates that as changes increase, the number of classes that depend on some high responsibility classes also grow. This clearly shows the ripple effect that can be created when changes are applied to those high responsibility classes. This shows maintainability of such a system will be low and hence changeability becomes low. The maintainability of a system is improved when changes can easily be made without propagating to other parts of the system. The larger the number of dependents of a class, the higher the likelihood of ripples effects when a change is implemented to the base class. Therefore, changeability declines with increasing Ca.

In the literature some researchers consider CBO and Ce as synonymous. As Ce depicts one direction of dependency, this interchangeability is not appropriate. Still others consider CBO as the combination of Ce and Ca. Again here, since CBO also refers to dependencies through inheritance and other dependencies, it is more than a combination of Ca and Ce. It is for this reason that some researchers (e.g., chaumun et al. (2000)) explicitly state dependencies such as CBO_NA (CBO No Ancestor).

5.2 Threats to Vailidity

We have demonstrated in our case study that apart from the common metrics such as CBO and WMC, coupling metrics such as Ce and Ca can be considered as good predictors of changeability of a system. However, these results should be taken cautiously as there are some issues that might have affected the validity of our case study results. First, as a case study of one open source system, the results will be difficult to generalize. Therefore, a significant number of systems should be assessed in relation to the metrics investigated in this study. Nevertheless, the data collected is from a historical maintenance data from the SVN repository of OpenBravoPOS. In addition, this system is being used by many businesses around the world which makes the change data a real data that has been collected from its change history. In addition, even though the size of OpenBravoPOS is 119,269 LOC, we considered only the Java code (62,417 LOC) for analysis which is not large. Therefore, generalization from the result of such analysis may be difficult.

Second, we considered the maintenance tasks applied to release 2.10 which resulted in release 2.20 with a total of 62 maintenance tasks. Whether these changes are representatives of the evolution of OpenBravoPOS cannot be easily predicted. However, we plan to expand our investigation up to the latest release of the product and also compare the pattern of change from one release to the next.

Third, the coupling metrics investigated are few. There are many coupling metrics proposed in the literature; however, some are redundant. Therefore, it is important to investigate as many coupling metrics as possible by carefully removing redundant metrics. In that case, we may find a different set of coupling metric as a better indicator of changeability. This will be one of our future works.

6. Conclusions

In this paper we presented the results of our case study which focused on investigating the capability of four metrics in assessing changeability. For this purpose, we collected historical maintenance data of evolving open source software OpenBravoPOS. Specifically, our study focused on analyzing the maintenance tasks applied to release 2.10 of the product to upgrade it to release 2.20. In total, we analyzed 62 maintenance tasks and their impact on the parts of the system.

Our case study results show that CBO, Ce, WMC, and Ca are good indicators of changeability of the system considered. Even though Ce and Ca are closely related to CBO, Ce showed a stronger correlation to maintenance tasks as compared to Ca. A study by Dubey and Rana (2011) determined that maintainability of a system is inversely proportional to the CK metrics which includes CBO and WMC. Therefore, higher values of CBO and WMC indicate low maintainability. In other words, as CBO and WMC values increase, the maintainability of a system decreases. Hence, in our case, the strong correlation of CBO, Ce, WMC and Ca with maintenance tasks indicates that as the values of these metrics increase, the changeability of the system decreases. From this we can conclude that CBO and Ce can be good indicators of changeability as are WMC and Ca. For a system to be changeable the values of the CBO, Ce, WMC, and Ca metrics should remain low.

In our future work, we would like to investigate two issues. First, we plan to investigate all relevant coupling metrics to determine the most appropriate metrics for changeability. Second, we plan to extend our investigation to the other sub-characteristics of maintainability (e.g., analyzability, adaptability, and stability) to determine the most appropriate metrics for these sub-characteristics of maintainability.

References

Ajrnal, C. M., Kabaili, H., Keller, R. K., Lustman, F., & Saint-Denis, G. (2000). Design properties and object-oriented software changeability. *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering, 2000* (pp. 45-54). http://dx.doi.org/10.1109/CSMR.2000.827305

Arisholm, E. (2002). Dynamic coupling measures for object-oriented software. *Eighth IEEE Symposium on Software Metrics* (pp. 33-42). http://dx.doi.org/10.1109/METRIC.2002.1011323

Arisholm, E. (2006). Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology, 48*(11), 1046-1055. http://dx.doi.org/10.1016/j.infsof.2006.01.002

Bohner, S., & Arnold, R. (1996). Software Change Impact Analysis: IEEE Computer Society Press.

Broy, M., Deissenboeck, F., & Pizka, M. (2006). Demystifying Maintainability. *Proceedings of the 2006 international workshop on Software quality* (pp. 21-26). http://dx.doi.org/10.1145/1137702.1137708

Chaumun, M. A., Kabaili, H., Keller, R. K., & Lustman, F. (1999). A change impact model for changeability assessment in object-oriented software systems. *Proceedings of the Third European Conference on Software Maintenance and Reengineering* (pp. 130-138). http://dx.doi.org/10.1109/CSMR.1999.756690

Chaumun, M. A., Kabaili, H., Keller, R. K., Lustman, F., & Saint-Denis, G. (2000). Design properties and object-oriented software changeability. *Fourth European Conference on Software Maintenance and Reengineering* (pp. 45-54). http://dx.doi.org/10.1109/CSMR.2000.827305

Dagpinar, M., & Jahnke, J. H. (2003). Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison. *Proceedings of the 10th Working Conference on Reverse Engineering* (pp. 155-164). http://dx.doi.org/10.1109/WCRE.2003.1287246

Dubey, S. K., & Rana, A. (2011). Assessment of Maintainability Metrics for Object-oriented Software System. *ACM SIGSOFT Software Engineering Notes, 36*(5), 1-7. http://dx.doi.org/10.1145/2020976.2020983

Eaddy, M., Zimmermann, T., Sherwood, K. D., Garg, V., Murphy, G. C., Nagappan, N., & Aho, A. V. (2008). Do Crosscutting Concerns Cause Defects? *IEEE Trans. Softw. Eng., 34*(4), 497-515. http://dx.doi.org/10.1109/TSE.2008.36

Emam, K. E., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software, 56*(1), 63-75. http://dx.doi.org/10.1016/S0164-1212(00)00086-8

Fluri, B. (2007). Assessing Changeability by Investigating the Propagation of Change Types. *29th International Conference on Software Engineering* (pp. 97-98). http://dx.doi.org/10.1109/ICSECOMPANION.2007.23

Grover, P. S., Kumar, R., & Kumar, A. (2008). Measuring changeability for generic aspect-oriented systems. SIGSOFT Softw. *Eng. Notes, 33*(6), 1-5. http://dx.doi.org/10.1145/1449603.1449610

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990.

Ingram, C., & Riddle, S. (2011). Linking software design metrics to component change-proneness. *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics* (pp. 31-37). http://dx.doi.org/10.1145/1985374.1985384

ISO/IEC9126-1. (2001). Software Engineering - Product Quality International Standard. Geneva.

Jalote, P. (2005). *An Integrated Approach to Software Engineering* (3rd ed.). Springer.

Li, B., Sun, X., Leung, H., & Zhang, S. (2012). A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability* (pp. 1-34). http://dx.doi.org/10.1002/stvr.1475

OpenBravoPOS. (2008). *OpenBravoPOS*. Retrieved November 1, 2012, from forge.openbravo.com/projects/openbravopos

Poshyvanyk, D., & Marcus, A. (2006). The Conceptual Coupling Metrics for Object-Oriented Systems. *Proceedings of the 22nd IEEE International Conference on Software Maintenance* (pp. 469-478). http://dx.doi.org/10.1109/ICSM.2006.67

Riaz, M., Mendes, E., & Tempero, E. (2009). A Systematic Review of Software Maintainability Prediction and Metrics. *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement* (pp. 367-377). http://dx.doi.org/10.1109/ESEM.2009.5314233

Robillard, M. (2005). *FEAT: An Eclipse Plug-in for Locating, Describing, and Analyzing Concerns in Source Code*. Retrieved February 19, 2012, from http://www.cs.mcgill.ca/~swevo/feat/

Sahraoui, H. A., Godin, R., & Miceli, T. (2000). Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation? *Proceedings of the International Conference on Software Maintenance* (pp. 154-162). http://dx.doi.org/10.1109/ICSM.2000.883034

Samoladas, I., Stamelos, I., Angelis, L., & Oikonomou, A. (2004). Open Source Software Development Should Strive for Even Greater Code Maintainability. *Communications of the ACM, 47*(10), 83-87. http://dx.doi.org/10.1145/1022594.1022598

Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., & Offutt, A. J. (2002). Maintainability of the Linux Kernel. *IEEE Software, 149*(1), 18-23. http://dx.doi.org/10.1049/ip-sen:20020198

Sun, X., Li, B., & Zhang, Q. (2012). A Change Proposal Driven Approach for Changeability Assessment Using FCA-based Impact Analysis. *Proceedings of the 36th International Conference on Computer Software and Applications* (pp. 328-333). http://dx.doi.org/10.1109/COMPSAC.2012.44

Wilkie, F. G., & Kitchenham, B. A. (2000). Coupling measures and change ripples in C++ application software. *Journal of Systems and Software, 52*(2-3), 157-164. http://dx.doi.org/10.1016/S0164-1212(99)00142-9

Yuming, Z., & Baowen, X. (2008). Predicting the Maintainability of Open Source Software Using Design Metrics. *Wuhan University Journal of Natural Sciences, 13*(1), 14-20. http://dx.doi.org/10.1007/s11859-008-0104-6