COMPARATIVE ANALYSIS OF SOFTWARE REUSABILITY ATTRIBUTES IN WEB AND MOBILE APPLICATIONS

BESHAR DHAYA NOR

A dissertation submitted in partial
fulfillment of the requirement for the award of the
Degree of Master of Computer Science (Software Engineering)

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

APRIL, 2015

ABSTRACT

Software reuse is an important approach to software engineering, where it aims to use previous software components to create new software systems. Reusability minimizes work repetition, development time, cost, efforts, and increases systems reliability. Reusability measurements help developers to provide the right metrics for measuring the reusability attributes and to identify reusable components among the wealth of existing programs. The main problem encountered in software reuse is the proper selection of the right software component for reuse due to similarity between the desired functionality and the function of the retrieved software component. In addition, it is difficult to define the right metrics that capture important quality attributes of a given class. This research aims to identify and measure the attributes that affect the software components reusability in two open source web and mobile applications. It also aims to compare the usage rate of reusability components in these applications to decide their ability to reuse. Four attributes were selected due to their impacts on reusability namely flexibility, portability, variability and understandability. Five metrics were identified to measure these attributes based on specified formulas. The metrics are Coupling Between Object (CBO), Lack Of Cohesion (LCOM), Depth Of Inheritance (DIT), Number Of Children (NOC) and Line Of Code (LOC). The research results indicate that the same attributes and metrics are suitable for measuring the reusability components in both applications. The comparison between the two applications for reuse indicates that the web application is more difficult for reuse than the mobile application.

ABSTRAK

Penggunaan semula perisian adalah pendekatan yang penting dalam kejuruteraan perisian, di mana ianya bertujuan untuk menggunakan komponen-komponen perisian yang dibangunkan sebelum ini untuk membina sistem baru. Penggunaan semula akan meminimumkan kerja yang berulang-ulang, masa pembinaan, kos, usaha, dan meningkatkan kebolehpercayaan sistem. Mengukur penggunaan-semula membantu pembangun sistem mendapatkan metrik yang betul dalam mengukur ciri-ciri penggunaan-semula dan seterusnya mengenalpasti komponen-komponen yang boleh diguna semula di dalam program-program sedia ada. Masalah utama yang berlaku di dalam kaedah penggunaan-semula adalah pemilihan komponen perisian yang terbaik dan bersesuaian disebabkan persamaan di antara fungsi-fungsi yang diingini dengan fungsi-fungsi yang telah sedia ada di dalam komponen perisian yang ingin diambil fungsinya. Tambahan lagi, adalah sukar untuk menentukan metrik yang betul yang dapat mengambil ciri-ciri yang terbaik di dalam kelas yang diberikan. Penyelidikan ini memfokuskan untuk mengenal pasti dan mengukur ciri-ciri yang mempengaruhi penggunaan semula komponen-komponen perisian di dalam dua sumber terbuka iaitu aplikasi laman web dan aplikasi mudah alih. Ia juga memfokuskan untuk membuat perbandingan berkenaan kadar penggunaan komponen-komponen yang digunakan semula dalam dua aplikasi ini bagi menentukan keupayaan ia untuk digunakan semula. Empat ciri telah dipilih disebabkan impak ciri tersebut kepada penggunaan-semula iaitu fleksibiliti, mudah alih, kepelbagaian dan kebolehan memahami. Lima metrik telah dikenalpasti untuk mengukur sifat-sifat ini berdasarkan formula yang khusus. Metrik tersebut adalah CBO, LCOM, DIT, NOC dan LOC. Hasil penyelidikan ini menunjukkan ciri-ciri dan metrik tersebut adalah sesuai untuk mengukur komponen penggunaan semula dalam kedua-dua aplikasi. Perbandingan tersebut menunjukkan aplikasi web lebih sukar untuk diguna semula berbanding aplikasi mudah alih.

CONTENTS

TITLE		**i**
DECLARATION		**ii**
DEDICATION		**iii**
ACKNOWLEDGEMENT		**iv**
ABSTRACT		**v**
ABSTRAK		**vi**
CONTENTS		**vii**
LIST OF TABLES		**x**
LIST OF FIGURES		**xi**
CHAPTER 1	**INTRODUCTION**	**1**
1.1	Introducation	1
1.2	Problem Statements	2
1.3	Research Questions	3
1.4	Research Objectives	3
1.5	Research Significance	4
1.6	Research Scope	4
1.7	Dissertation Outlines	5
CHAPTER 2	**LITERATURE REVIEW**	**6**
2.1	Introduction	6
2.2	Software Quality	6
2.3	Software Reusability	7
2.3.1	Reuse Benefits	9

2.3.2	Reusability in Component	10
2.3.3	Software Component	12
2.4	Reusability Attributes	12
2.5	Common Attributes Definitions	14
2.5.1	Flexibility	14
2.5.2	Maintainability	14
2.5.3	Portability	14
2.5.4	Variability	14
2.5.5	Understandability	15
2.5.6	Size	15
2.5.7	Complexity	15
2.5.8	Scope coverage	15
2.5.9	Availability	15
2.6	Reusability Metrics	16
2.6.1	Object Oriented Metrics	17
2.6.2	Component Based Metrics	18
2.7	Common Metrics	20
2.7.1	Lines of Code (LOC)	20
2.7.2	Depth of Inheritance Tree (DIT)	20
2.7.3	Number of Children (NOC)	20
2.7.4	Coupling (CBO)	21
2.7.5	Cohesion (LCOM)	21
2.7.6	Number of Methods (NOM)	21
2.8	Web Applications	22
2.8.1	Design	22
2.8.2	Reuse	23
2.9	Mobile Applications	24
2.9.1	Design	25
2.9.2	Reuse	27
2.10	Related Work	28
2.11	Summary	31
CHAPTER 3	**RESEARCH METHODOLOGY**	**32**
3.1	Introducation	32

3.2	Research Design	32
3.3	Phase1	34
3.3.1	Coupling and Cohesion	35
3.3.2	Depth of Inheritance Tree	35
3.3.3	Number of Children	36
3.3.4	Lines of Code	36
3.4	Phase2	36
3.4.1	Selected Applications	36
3.4.2	Metrics Calculation	37
3.5	Phase3	37
3.6	Summary	37
CHAPTER 4	**RESULTS AND ANALYSIS**	**39**
4.1	Introduction	39
4.2	Practical Approach to Evaluate Reusability	39
4.3	Web Application	40
4.3.1	Metrics	40
4.3.2	Attributes Measurement	42
4.4	Mobile Application	45
4.4.1	Metrics	45
4.4.2	Attributes Measurement	47
4.5	Comparison Results	48
4.6	Summary	53
CHAPTER 5	**CONCLUSIONS AND FUTURE WORKS**	**55**
5.1	Introduction	55
5.2	Research Findings	55
5.3	Future Works	56
	REFERENCES	**57**
	VITA	**123**

LIST OF TABLES

2.1	Comparison of different metrics	19
2.2	Effect of metrics on reusability	22
2.3	Related work on measuring reusability	31
3.1	Reusability attributes and metrics	34
3.2	Metrics calculations	37
4.1	Result of all classes in web application	42
4.2	Result of all classes in mobile application	47
4.3	CBO metric for both applications	48
4.4	DIT metric for both applications	49
4.5	LOC metric for both applications	50
4.6	LCOM metric for both applications	51
4.7	NOC metric for both applications	52
4.8	Effects of attributes and metrics on reusability	53

LIST OF FIGURES

3.1	Research procedure	33
4.1	Class DAOfactory	41
4.2	Class Instructionactivity	45
4.3	CBO representation	48
4.4	DIT representation	49
4.5	LOC representation	50
4.6	LCOM representation	51
4.7	NOC representation	52

LIST OF SYMBOLS AND ABBREVIATIONS

CBO	-	Coupling Between Object.
LCOM	-	Lack of Cohesion.
NOC	-	Number of Children.
LOC	-	Line of Codes.
DIT	-	Depth of Inheritance Tree.
NOM	-	Number of Methods.
CBD	-	Component-Based Development.
WMC	-	Weighted Methods Per Class.
RFC	-	Response For Class.
AR	-	Average Degree of Reusability.
SBRM	-	Specialize Class to Base Class Reusable Metrics.
CC	-	Cyclomatic Complexity.
CPD	-	Component Packing Density.
CID	-	Component Interaction Density.
ROI	-	Relevance of Identifiers.
CIC	-	Correlation Identifier's Comments.

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Web Application	64
B	Mobile Application	107

CHAPTER 1

INTRODUCTION

1.1 Introduction

Software has become critical to advancement in almost all areas of human endeavor. There are serious problems in the cost, timeliness, maintenance and quality of many software products. Software engineering has objectives of solving these problems by producing good quality, maintainable software, on time, within budget (Singaravel *et al.,* 2010). Generally, a software-development work requires various kinds of resources to complete the project on time. The ability of some functions or packages to be reused in the project will help to evolve the software instead of developing the module from the scratch (Babu & Srivatsa, 2009). Software reuse is an important approach to software engineering. It is the process of building applications that made use of formerly developed software component (Bhanu, 2014).

Software reusability is an important aspect of the software-development process, where it can use previous components to create new software systems. The cost to develop a new system from scratch is significant. This has made custom software development very expensive. It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. This concept is almost universally accepted because of the obvious fact that a product will work properly if it has already worked before (Sharma *et al.,* 2009). The idea of software reuse appeared in 1968, opening new horizons for software design (Sandhu *et al.,* 2010), and have been promoted in recent years. The software-development community is gradually drifting toward promoting software reuse to develop any new software system virtually from the existing systems (Gil, 2006). Reusability is

as the degree to which a component can be reused and minimizes repetition of work, development time, cost, efforts, and increases system reliability (AL-Badarenn *et al.*, 2011). It also improves the maintainability and portability of the system (Sharma *et al.*, 2009). As an example of the role of reusability in reducing the cost of the software development, (Agrawal & Patel, 2012) stated that the U.S. Department of defense only could be saving $300 million annually by increasing its level of reuse by as little as 1%. In addition, using a software reuse concept in Missile Systems Division increased the productivity by 50%, and using reusable modules to develop the prototype of a force fusion system reduced 20% of the development time of the estimated time for developing the new system.

A good software reuse process can facilitate the increase of productivity of program design and development, reliability of software products, and the decrease of costs and implementation time. Technologies would enable software companies to build specialized components that can be sold to systems integrators and custom builders, who would combine them with other, largely purchased, off-the- shelf components to create high-quality custom applications. An established software components industry would provide a rapid and responsive channel for marketing software innovations, while constantly improving quality, reliability, and capability (Sandhu *et al.*, 2010).

To assess various properties of components to choose the right components and reuse them correctly, metrics is required (Monga *et al.*, 2014). The measurements of the reusability help developers to control the level of the reuse and providing the metrics for identifying one of the important quality property's reusability. The measurement may help us not only to learn how to build reusable components, but also to identify reusable components among the wealth of existing programs (Agrawal & Patel, 2012).

1.2 Problem Statement

At present, reuse includes different approaches, such as reusing components developed in-house, reusing of commercial-off-the-shelf (COTS) or open-source software (OSS) components (Mohagheghi & Conradi, 2008). Software reuse aims to use existing components to build new components or products. Software may be

source code or executable, design templates, software architectures, or any other asset. Due to increasing number of components in the market, it becomes necessary to qualify the reusability of these components to define the effective ones for reuse (Sharma *et al.,* 2009). However, it is a challenge to find the right reusable artifacts from huge components. In addition, measuring software reusability attributes is a difficult activity for both developers and managers (Kumari, 2011, Bauer *et al.,* 2014). The problems of selecting the proper software component for reuse (Otis *et al.,* 2013) are related not only to the similarity between the functions delivered by the retrieved software component, but also to the effort needed to modify the chosen component to accommodate the desired functionality. That is why it is so important to have efficient reuse metrics that assists to use a given software component in new environment (Sandh *et al.,* 2010).

This dissertation selects some attributes that affect software reusability and identify a set of metrics for measuring the reusability of components in two open source object-oriented web and mobile applications. In addition, their usage rate in these applications is compared to define the possibility of reusing the components and the applications. This can help developers in the selection process of the right components for reuse in other software environments and shorten the effort, cost and time. Moreover, it helps to acquire sound knowledge of the required metrics.

1.3 Research Questions

The research problem raises the following questions:

(i) What are the attributes that affect the software component reusability?

(ii) How to measure the reusability attributes of these components?

(iii) Do the web and mobile applications use the same attributes at the same rate?

1.4 Research Objectives

(i) To identify attributes that affect reusability of software components in web and mobile applications.

(ii) To measure the attributes reusability in both applications with proper metrics.

(iii) To perform comparative analysis between the usage rates of component reusability in both applications that assists in selection of components for reuse.

1.5 Research Significance

Selection of suitable components is necessary to achieve objectives for improving product quality within time and budget constraints. Identifying the significant attributes and the proper metrics to measure the quality and efficiency of software components to be reused will help in building high quality and reliable software with reduced cost and time. They help in project estimation and progress monitoring, evaluation of work products, process improvement, and experimental validation of best practices.

1.6 Research Scope

This research aims to measure the attributes that affect the reusability of components of two object oriented open sources in mobile and web software applications, and compare the usage rate of component reusability in these applications, where there are no standard limits or specified range of the metrics value except high and low criterion (Amin *et al.* 2011, Taibi, 2014, Agrawal & Patel, 2012, Monga *et al.* 2014, Taibi, 2013, Dubey & Rana, 2010, Gui, 2009). Four reusability attributes of components will be measured according to measurement metrics to define their rate in each application. This will lead to determining the component's ability to be reused in building new software application. The focus of this research is as follows:

(i) Two open source component based applications related to exam system; one mobile application (Dice, 2014) and another web application (Osama, 2012) are chosen.

(ii) The following attributes that affect their component reusability and their metrics are selected:

- Flexibility: Coupling between object (CBO) and lack of Cohesion (LCOM).
- Variability: Number of Children (NOC).
- Understandability: Size of Codes (LOC).
- Portability: Depth of inheritance tree (DIT).

(iii) Compare the usage rate of component reusability in the two applications.

1.7 Dissertation Outlines

This research is organized in five chapters. Chapter 2 reviews the literature, including the notion of web and mobile applications, their reusability attributes and metrics. Chapter 3 explains the methodology and the attributes that affect the reusability and their metrics. Chapter 4 shows the results, which includes the reusability attribute measurements. It also includes a comparison of the usage rate of each reusability component in two open source object-oriented web and mobile applications. The conclusions and future work are presented in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Software Engineering aims for quality software producing strategies. In which software development life cycle involves sequences of different activities during the development process (Singaravel *et al*., 2010). Software reuse is the use of existing software artifacts to create new software (Singh *et al*., 2010). It plays an important role in increasing quality of software products (Gandhi & Bhatia, 2011).

Huge web and mobile applications are created to perform different tasks on different platforms. The applications' construction is commonly based on components based or objects oriented. So same attributes characterize them. Researchers used much metrics to measure various reusability attributes for all applications. This chapter reviews the previous studies and researches and introduces a brief knowledge about web and mobile applications. It also defines software reusability concept, definition, benefits and reusability metrics.

2.2 Software Quality

Developing a good software system is a very challenging task. To produce a good software product, numerous measures for software quality attributes need to be taken into account (Iqbaland & Qureshi, 2012, Suri & Garg, 2009) System complication dimensions play a vital role in controlling and supervision of software quality because it normally affects the software quality attributes like software reliability,

software testability and software maintainability, indicated that reusability is one of the attributes that determine the quality into the software.

Reusability is the key paradigm for increasing software quality in software development. It is an important area of software engineering research that promises significant improvements in software productivity and quality (Singh *et al*., 2010).

Pylkki (2013) the main objective of reuse is to increase software quality and productivity. Reusable assets are not limited to the source code, but can also include documentation or organizational practices. Modern programming languages provide support for reusing large portions of source code with classes, modules and frameworks. Constructing systems out of ready-made components can manage complexity if the components have been clearly defined, standardized interfaces and connection methods.

Software reusability is an attribute that refers to the expected reuse potential as a software component (Sandhu *et al*., 2010). Since reusability is an attribute of software quality, hence can measure software quality by quantifying its reusability. Productivity, maintainability, portability and therefore, the overall quality into the product can be improved by software reusability (AL-Badareen *et al*., 2011).

2.3 Software Reusability

The idea of software reuse is not new (Hristov *et al.,* 2012). Its roots date back to 1968 when McIlroy has presented his seminal work on reusable components at the NATO Software Engineering Conference in Garmisch, Germany. However, there has only been limited practical experience with reuse until the late 1980s, when large-scale reuse programs were adopted by companies, mainly in the United States (e.g., by IBM and Hewlett Packard) and Japan (e.g., by Toshiba and Fujitsu). These efforts have also pushed forward the research in the 1990s, and in turn created a growing interest in systematic software reuse and reuse programs for organizations of that time.

Reuse is an act of synthesizing a solution for a problem based on predefined solutions to sub problems (Kumar, 2012). Software reusability refers to the probability of reuse of software. The ability to reuse relies in an essential way to the

ability to build larger things from smaller parts, and being able to identify commonalities among those parts.

Sandhu *et al.* (2010) Software reuse not only improves productivity but also has a positive impact upon the quality and maintainability of software products and enhances the reliability of a new software application.

AL-Badareen *et al*. (2011) The concept of the software reuse is not only applied to source code fragment, but can also mean all the information that is related to the product generating processes, including software requirements, analysis, design, and any information required by the developers to build a software.

Singaravel *et al*. (2010) The reusability of a piece of code does not mean that it can copy-paste the same code in many places within an application. A piece of reusable code means that the same code can be reused in different places without re-writing it. The metric of reusability is how many programs can reuse the piece of code without looking at the source code.

Software reuse in its most common form can be seen in the component based software development (Amin *et al.*, 2011), where it has been promoted in recent years. The software-development community is gradually drifting toward the promise of widespread software reuse, in which any new software system can be derived virtually from the existing systems. There are two approaches for reuse of code: develop the code from scratch or identify and extract the reusable code from already developed code (Manhas *et al*., 2010, Kumar, 2012).

Software reusability is defined by many researchers. Amin *et al*. (2011) defined it as the “characteristics of an asset that make it easy to use in different contexts, software systems, or in building different assets."

Babu and Srivatsa (2009) defined software reusability as the process of creating a software system from existing software assets rather than the building software system from scratch. Software reusability is the development of new software from the existing one.

Sagar *et al*. (2010) defined reusability as the degree to which a component can be reused, and reduces the software development cost by enabling less writing and more assemblies.

Sridhar *at el.* (2014) Software reuse is the process of building software applications that made use of formerly developed software components.

Singh *et al.* (2010) defined reusability as the use of engineering knowledge or artifacts from existing software components to build a new system. Reusability is the key paradigm for increasing software quality to the software development.

Agrawal & Patel (2012) defined the reusability as one of the quality attributes that has prime importance in object oriented software development as reusability leads to increase in developer productivity, reduce a development cost as well as the time to market.

Sharma *et al.* (2009) defined reuse as the process of adapting a generalized component to various contexts of use. They indicated that the main idea of software reuse is to use previous software components to create new software programs. Thus, software reuse is software design, where previous components are the building blocks for the generation of new systems.

Trived & Kumar (2012) defined Software reuse in the process of implementing or updating software systems using existing software components.

Hristov *et al.* (2012) stated that it was important to distinguish between software reuse and reusability as the reuse is focused on the practice of reuse itself while the reusability tries to make the potential of artifacts for being reused measureable. Generally, the researchers use the two words synonymously.

2.3.1 Reuse Benefits

Software reuse is the improvement efforts of the productivity of the software because reuse can result in higher-quality software at a lower cost and delivered within a shorter time. It reused software is more accurate than new software because already it has been tried and tested in working system (Kumar, 2012).

Reusability reduces implementation time, increase the likelihood that prior testing and use eliminated bugs and localizes code modifications when a change in implementation is required (Kakkar *et al*., 2012).

Software reuse reduces development time, effort, cost and increase's productivity and quality. Studies in software engineering confirm these benefits (Amin *et al.*, 2011).

In general, the reuse of codes, components and other artifacts aims to (Trived & Kumar, 2012).

(i) Reduce time to market.
(ii) Reduce a development cost.
(iii) Improve the productivity of development teams.
(iv) Improve the predictability of the development process.
(v) Increase the quality and reliability of products.
(vi) Reduced overall process risk.

The main idea of software reuse is to use previous software components/artifacts to create new software systems. Therefore, it minimizes repetition of work, development time, cost and efforts and increase systems reliability. It also improves the portability and maintainability of the system (Sharma *et al.,* 2009).

2.3.2 Reusability in Component

In software industry, the evolution of reusability started from Object-oriented systems, then Component-based systems, and now it talks with Service-oriented systems (Karthikeyan & Geetha, 2012). In the case of component-based development, software reuse refers to the utilization of a software component within a product to be used in another product (Sharma *et al*., 2009). A reusable component can be seen as a box, which contains the code and the documentation. These boxes are defined as (Sharma *et al.,* 2009, Singh *et al.,* 2010).

(i) Black Box Reuse

In black box reuse, the re-user sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component.

(ii) Glass Box Reuse

In glass box reuse, the inside of the box can be seen as well as the outside, but it is not possible to touch the inside.

(iii) White Box Reuse

In white box reuse, it is possible to see and change the inside of the box as well as its interface. A white box can share its internal structure or implementation with another box through inheritance or delegation.

In component-based development, there are two broad reuse developments. One is the development of systems with reuse, and another is the development of components for reuse. In first case, application is developed by reusing several already-built-in components. These components have already been tested thoroughly, which will enhance the quality into the product and will save time and cost. Another approach is the development of components for reuse. Here, components are developed by keeping in mind the high reusability for this component. These components need to be compatible for a wide range of applications, developed in variety of languages for different platforms. One of the essential problems in software reuse is the retrieval and selection of suitable software components from a large library of components (Sharma *et al*., 2009).

In case of an object-oriented software programs, an object-oriented software system is a collection of classes, which abstract data types and templates in a way of making classes more abstract without actually knowing what data type will be handled by the operations on the class. With the help of template, a single class can be used to handle different types of data, and a single function can be used to accept different types of data, which makes the code easier to maintain and classes more reusable (Gandi *et al*., 2010).

An object-oriented system start by defining a class that contains related or similar attributes and methods. The classes are used as the basis for objects. A class is a template from which objects can be created. This set of objects shares a common structure, and a common behavior manifested by the set of methods. A method is an operation on an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. A high degree of inheritance is an indicator of system health. Strong coupling complicates a system, since a module is harder to understand, change, or correct if it is interrelated with other modules. The more independent a class, the easier it is to reuse it in another application. High cohesion indicates good class subdivision (Goel & Bhatia, 2013).

2.3.3 Software Component

Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch (Babu & Srivatsa, 2009). A component can be considered an independent replaceable part of the application that provides a clear distinct function. Reusable components can be requirements of specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software. All products resulting from the software-development life cycle have the potential for reuse. A software component can be a code block, module, function, class, control or the project or software itself (Trived & Kumar, 2012, Manhas *et al*., 2010). In the context of object orientation, a class can be said to be a component because it is the only unit of composition (Amin *et al.,* 2011). In the context of component-based software development, the component is a reusable piece of software (Capiluppi & Boldyreff, 2011).

2.4 Reusability Attributes

Software reusability is a significant aspect of the software-development process, where it is able to use previous software components to create new software systems. Software components have attributes that affect their reusability. These attributes are related to using a given software component in new environment and for another software system. The environmental attributes as well as attributes derived from the software should be considered, identified and measured. As the number of components available on the market increases, it is becoming more important to devise software metrics to qualify the various characteristics of components. Among several quality characteristics, the reusability is particularly important when reusing components (Sharma *et al.,* 2009).

Hristov *et al*. (2012) mentioned several characteristics of software to determine such factors are adaptability, complexity, compose-ability, maintainability, modularity, portability, programming language, quality, reliability, retrieve-ability, size and understandability. They indicated that most of the existing research is rather incoherent and only covers one or a few of these aspects so that to our knowledge,

there is no publication that has tried to bring all these aspects together in a single model.

Sandhu *et al.* (2010) indicated that metrics collectively determined reusability of a software component is still at its naive stage, and function that makes use of this metrics to find reusability of software components is still not clear.

Sagayaraj & Ganapathy (2011) mentioned some general reusability attributes include ease of understanding, functional completeness, reliability, good error and exception handling, information hiding, high cohesion and low coupling, portability and modularity.

Sharma *et al.* (2009) identified some attributes for predicting component reusability such as small size of code, simple structure and good documentation.Hence considered reusability is a measure of four factors on which the reusability of the component depends namely customizability, interface complexity, understandability, and portability. The customizability is defined as the ability to modify a component as per application requirement. It should be high, and it categorized from very low to very high where a number is assigned for each category. The interface complexity refers to the interface between component and applications, which acts as a primary source for understanding, use and implementation and finally maintenance of the component. The interface complexity should be as low as possible, and it is also categorized from very low to very high. The understandability refers to any documentation provides help to users, and it includes component manuals, demos, help system, and marketing information. Portability refers to the ability of a component to be transferred from one environment to another with little modification. The component should be easily and quickly portable.

Jatain & Gaur (2012) proposed a model based on four factors: Changeability, Interface Complexity, Understandability of Software and Documentation Quality. They used soft computing techniques, namely neuro-fuzzy approach to determine reusability of software components in existing systems as well as the reusable components. They claimed that neuro-fuzzy technique can be a powerful tool to tackle important problems in software engineering and can be further extended as software metric model.

Taibi (2014) indicated that complexity, modularity and understandability enhance the quality of object-oriented program source code and potentially improve its reusability.

2.5 Common Attributes Definitions

2.5.1 Flexibility

It is the ease of modification system or component for use in applications or environments other than that it was designed for it (Amin *et al.*, 2011, Monga *et al.*, 2014).

2.5.2 Maintainability

It is the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment (Monga *et al.*, 2014).

Maintainability is related to reusing in terms of error tracking and debugging. If the component is maintainable, it is more likely to be reused (Amin *et al.*, 2011).

2.5.3 Portability

It is the ease with which a system or component can be transferred from one hardware or software environment to another. Portability is considered as a factor in the sense that a cohesive component is more portable. The portability of a component depends on its independence, i.e. the ability of the component to perform its functionality without external support (Amin *et al*., 2011, Monga *et al.*, 2014).

2.5.4 Variability

It can be defined as the configure ability of a component, that it can be configured in

multiple configurations. Variability decreases understandability (Amin *et al.*, 2011).

2.5.5 Understandability

It is the ease with which a system can be comprehended at both the system organizational and detailed statement levels. The understandability attribute is also related to the maintainability of the component, a component that is easy to understand is easy to maintain (Amin *et al.*, 2011). It also can be defined as the degree of knowing how and what things are working and interacting in the system (Monga *et al.*, 2014).

2.5.6 Size

Size may include only the lines of code in a program, or it may include the lines of code along with commented line of code (Monga *et al*., 2014). It is the length of the software (Amin *et al*., 2011).

2.5.7 Complexity

It refers to the difficulty in understanding the design and implementation of software. A complex software or component is difficult to be reused (Monga *et al.*, 2014).

2.5.8 Scope coverage

It is the attribute that measures the number of features provided by the component against the total number of features in the SPL scope (Amin *et al.*, 2011).

2.5.9 Availability

It refers to how easy and fast (or hard and slow) is to retrieve a software component (Hristov *et al*., 2012).

2.6 Reusability Metrics

The potential benefits of software reuse and the maturity of reusability concepts lead us to think about how the author might measure it (Amin *et al*. 2011). The concept of reuse is very popular today because it saves time, cost, and effort to develop a software system. Thus, it is necessary to have an effective and efficient method to measure software component reusability. Measuring reusability is not a straightforward process due to the variety of metrics and qualities linked to software reuse and the lack of comprehensive empirical studies to support the proposed metrics or models (Taibi, 2013,Taibi, 2014).

Software metrics is units of measurement used to measure different attributes of a software product, and process. Metrics plays a very important role to develop good-quality software (Kumari & Bhasin, 2011).

Researchers used much metrics to measure various reusability attributes for all applications. The software metrics is useful in helping software developers to develop effective software reuse requires that the users of the system have access to appropriate components. The user must access these components accurately and quickly, and be able to modify them if necessary. Various attributes, which ascertain the quality into the software, include maintainability, defect density, fault proneness, normalized rework, comprehensibility, reusability, etc. At present, the requirement is to associate the reusability attributes with the metrics and how this metrics collectively ascertains the reusability of the software component (Suri and Garg, 2009).

Metrics should be identified to measure these attributes. It relates to a defined measurement approach and a measurement scale. A metric is expressed in units, and can be defined for more than one attribute. The primary aims for reusing metrics are (Patwa and Malviya, 2012):

(i) To provide realistic measures of reuse

(ii) To estimate the benefits of reuse: The metrics is useful in finding the estimates of the benefits from specific factor, i.e. reuse and testing. Exact values matter less than reasonable estimates.

(iii) To provide feedback to developers and management:

Developers need a defined method to measure and report what they have done.

(iv) To give simple, easy-to-understand values.

(v) To yield consistent, repeatable values independent of who calculates them and proves them mathematically and / or axiomatically sound.

There are a number of metrics available for measuring the reusability for object-oriented systems. They focus upon the object structure, which reflects on each individual entity such as methods and classes, and on the external attributes that measure the interaction among entities such as coupling &inheritance (Sharma *et al.*, 2009). Some of these object-oriented metrics can be used in CBD (Sagar *et al*., 2010). However, there are some difficulties in applying existing object-oriented metrics into the component development and CBSD because these metrics requires analysis of source code. Object- oriented (OO) metrics cannot be used to measure the component's quality. The reason may be that in OO development, reuse is only limited up to class level and within the same application, while in CBSD, the reuse is able even the whole component and also in multiple applications (Sharma *et al*., 2009).

2.6.1 Object-oriented (OO) Metrics

OO design techniques have become one of the most powerful mechanisms to develop an efficient software system because it promotes better design and views a software system as a set of interacting objects. OO software can play an important role in reusability for software applications and development (Dubey& Rana, 2010).

With the advent of an OO approach, specific measures were introduced to assess the quality of OO software systems. The rationale behind OO metrics is that a good OO design must keep complexity low, and this can be accomplished by reducing coupling and increasing cohesion. The first serious attempt in this direction was the metrics suite by Chidamber and Kemerer (CK), which became the most popular OO metrics suite. This metrics is weighted methods per class (WMC), depth of the inheritance tree (DIT), number of children (NOC), coupling between object classes (CBO), response to a class (RFC) and lack of cohesion in methods (LCOM), (Concas *et al.*, 2010).

The application of object-oriented technology produces well-structured software systems with comprehensible architectures, which are easy to test, maintain, and extend. However, the OO approach does not ensure the production of software with high quality nor avoid errors introduced by programmers at both development and maintenance phases. Therefore, various OO metrics is proposed in the literature as a means of determining whether the investigated software hold desired OO design properties such as coupling, complexity, cohesion, and inheritance to improve the quality into the software. Many of these measures have been proposed by different researchers and practitioners (Cheikhi *at el*, 2014).

Patwa and Malviya (2012) proposed much software metrics to measure the benefits of reuse within OO system. These systems are reusability of a class in a System (RCS), average degree of reusability (AR) metrics and specialize class to base class reusable metrics (SBRM).

Gandhi & Bhatia (2010) used the same metrics of CK (1994) and stated that the most significant metrics for reusability among those metric is DIT, which indicates the length of inheritance and NOC, which indicates the width.

Kaur and Singh (2013) mentioned several complexity metrics for OO program reusability. These metrics includes WMC, DIT, RFC, CBO, LCOM, NOC, LOC, and cyclomatic complexity (CC).

Goel & Bhatia (2013) used different reusability metrics to evaluate three features of OO program, namely multilevel inheritance, multiple inheritances and hierarchical inheritance. The metrics is DIT, NOC, CBO, LCOM, WMC, and RFC. Their results have shown that multilevel inheritance has more impact on reusability.

2.6.2 Component-based Metrics

Component-based development (CBD) is the process of assembling existing software components into an application such that they satisfy a predefined functionality. CBD reduces development time, effort; cost (Sagar *et al.*, 2010). It is necessary to measure the software complexity in each development approach because it affects many other aspects of software like development effort, cost, testability, maintainability, etc. So many metrics has been proposed for measuring software complexity (Kaur and Singh, 2013).

Narasimhan *et al.* (2009) conducted a comparison among different metrics to select a particular type of metrics based on reusability, complexity, size, testing time and maintenance. The metrics values are compared using benchmark software programs. The metrics in table 2.1 measure the reusability of a component. A high LCOM, NOC, and DIT imply that the corresponding components are highly reusable. A high CID, CPD, WMC, CSC, and CBO, implies that the corresponding components are less reusable. A low CRIT Size, CRIT Link implies that the corresponding components are highly reusable. It is noted that a component is considered a good one if it is highly reusable.

Table 2.1: Comparison of Different Metrics (Narasimhan *et al.*, 2009).

Metrics	**Author (Year)**	**Strengths & Limitations**
WMC, RFC, LCOM, CBO, DIT, NOC	Chidamber & Kemerer (1994).	Broad indicator, but lack of specificity.
CPC, CSC, CDC, CCC	Cho & Kim (2001).	Narrow indicator.
CPD, CID, CIID, COID, CAID, CRIT, CRIT bridge, CRIT, ANAC, ACD, AACD.	Narasimhan & Hendradjaya (2007).	Covers a broad set of issues.

Kaur and Singh (2013) mentioned existing complexity metrics for component based. The metrics included component packing density (CPD), component interaction density (CID), component incoming interaction density (CIID), component outgoing interaction density (COID), component average interaction density(CAID), link criticality metric (CRITlink), bridge criticality metric (CRITbridge), inheritance criticality metric (CRITinheritance), size criticality metric (CRITsize), and Criticality Metric. The researchers state that most of the existing metrics are applicable to small programs or components, while the objective of having metrics is to test the behavior, reusability, and reliability of the components when placed in a large system. Since measuring the black box component complexity

during component selection is still a difficult task, they proposed a metric named Interface Complexity Metric for Black Box components, which is based on component interface specification. This metric helps an application developer in selecting a less-complex and more-reusable component during the selection of components for CBSD. This facilitates in reducing the integration and testing effort.

2.7 Common Metrics

Literature contains much metrics for reusability attributes as explained in previous sections. Clear and brief explanations of common metrics, especially the ones used for this research are provided as follows.

2.7.1 Lines of Code (LOC)

This metrics is applied for measuring the size of the program by considering the number lines into a program. LOC counts all lines like as source line and the number of statements, the number of comment lines and the number of blank lines (Agrawal and Patel, 2012). It is generally used to calculate the size of software or its components by calculating the total number of lines in it (Monga *et al.*, 2014).

2.7.2 Depth of Inheritance Tree (DIT)

This metric is applied for measuring the inheritance complexity of the programs. DIT is the Maximum depth from the root node of a tree to special node. Here, class is represented as a node. Deeper node on the tree accepts more the methods because they inherit and the more classes in the tree, and it makes the class more complex (Agrawal and Patel, 2012). A high DIT value is known to increase the number of faults (Monga *et al.*, 2014).

2.7.3 Number Of Children (NOC)

NOC is applied when there are many sub-classes of the particular class in the hierarchy within the class exist. When children as a class are more, then it requires

more testing because super class may be misused (Agrawal and Patel, 2012). It is the measure that counts the children as a class. A large number of children mean that the functionality to the class is reused through inheritance (Amin *et al.*, 2011).

2.7.4 Coupling Between Object (CBO)

Coupling is also called dependency. It is the level of dependency of one module to another. Coupling is one important component, which help to determine the quality to the design or software. Good program design is to achieve low coupling (Agrawal and Patel, 2012). Coupling means links or dependency of a class to be other. High CBO means low reusability (Monga *et al.*, 2014). Gui (2009) defined Coupling as the extent to which the various sub-components interact. If they are interdependent, then changes to one are likely to have significant effects of the behavior of others. Hence, loose coupling between its sub-components is a desirable characteristic.

2.7.5 Lack Of Cohesion (LCOM)

Cohesive indicates that a certain class performs a set of closely related to actions. An LCOM means that a class is performing various unrelated tasks. Principles of OO demand low coupling between modules and high cohesion of the module (Agrawal and Patel, 2012). The cohesion ensures that a specific class is not or least dependent on some other method or class (Monga *et al*., 2014).

2.7.6 Number Of Methods (NOM)

This metric gives the average number of operation per class (Monga *et al.*, 2014). It is an indicator of the size of a class (Amin *et al.*, 2011).

Monga *et al.* (2014) had summarized the effects of the metrics on reusability as shown in table 2.2.

Table 2.2: Effect of Metrics on Reusability (Monga *et al*., 2014).

Metric Name	Metric Value	Reusability Value
LOC	Increase	Decrease
DIT	Increase	Decrease
NOC	Increase	Decrease
LCOM	Decrease	Increase
CBO	Increase	Decrease
NOM	Increase	Increase

2.8 Web Applications

Web applications refer to applications designed to work on desktop computer browsers. Essentially, they work with devices with a browser. They can also work on mobile devices, being given the condition that they do not rely on specific browser features that are unavailable on most mobile devices (Serran *et al.*, 2013).

2.8.1 Design

Building complex Web application is a time-consuming task as they must provide navigational access to critical information resources, not only allowing the user to browse through the potentially large universe of information but also to operate it.

Most methodologies of Web application design formalize the design of a Web application by three models: the application (or content) model, the navigation model, and the presentation model. The application model defines the contents of the application and its behavior. The navigation model defines the information units of consumption (nodes) for the user and the navigation paths (links) between units. The presentation model defines the abstract user interface. There are two levels of abstractions or approaches, in which the design of a Web application may be improved while preserving its functionality. An approach is to apply changes in the code level in order to increase maintainability and extensibility of the application. A second approach to design improvement of a web application is to apply changes at the model level. For OO web design methods like OOHDM, the application model of

a web application is an OO model and as such. Changes at this level are described by model refactoring, which affects mainly on internal qualities, such as maintainability (Garrido *et al.*, 2009).

At present, most web applications are mostly designed with multiple tiers for flexibility and software reusability. It is difficult to model the behavior of multi-tier web applications because the workload is dynamic and unpredictable, and the resource demand in each tier is different. For example, the 3-tier web application architecture, which consists of presentation, application and data tiers, has been widely used (Huang *et al*., 2014).

2.8.2 Reuse

Reusability is important, especially in web application development because they need to be rapidly developed and frequently modified (Hokamura *et al*., 2010). Many industrial web application's software has been developed. Unfortunately, most of them were procedure-oriented, thus making them unsuitable for reuse and customization effectively as well as becoming more and more complicated. Considering this, efforts have been made to push legacy system software into the new OO technology development. There are many repeated works in this development, particularly in the design phase. An approach is needed to achieve reusability, extensibility and reliability in web application development; only then, web engineers/developers can reuse design as well as implementation. The need for software reuse has become evident because complex software remains difficult to implement, expensive to develop and risky to maintain. The idea behind reuse is not to develop anything that already exists, but just reuses it. This will lead to shortened development time, reduced complexity, increased productivity, extensibility and reliability of web applications (Nuruzzamanet *et al.,* 2013). He has offered a novel solution to produce high-quality web applications within a shortest development timeframe through the means of customization, reusability, extensibility and flexibility. They conducted a comprehensive evaluation on the proposed OO framework and emphasized the reuse of design, code and testing as a tool to uncover strengths and weaknesses of the OO framework for dynamic web engineering. There are several studies and open source frameworks for improving reusability of

components for web applications. However, the existing techniques are depended on specific frameworks or architecture, and the architecture, and the techniques require web applications to be implemented by the frameworks or architecture. Therefore, reusability brought by the techniques is restricted to limited web applications. There are common reusable functional such as access control, access analysis, and performance tuning for multiple web applications. Therefore, mechanism for implementing reusable components, which are available for multiple web applications is important. To achieve that, Hokamura *et al*. (2010) used a domain-specific aspect-oriented (AO) mechanism based upon an abstraction model common to all web applications. AO contributes to flexible mechanism by adding new functional for the base programs. They have indicated that the domain-specific AO mechanism is an effective platform to implement reusable functional common in many web applications.

Ghosheh *et al*.(2008) Proposed new metrics used for measuring the maintainability of web applications from class diagrams. The metrics is based on Web Application Extension (WAE) for UML to measure size, complexity, coupling and reusability.

2.9 Mobile Applications

They are applications developed to run on devices such as smart phones or tablets. Users typically access them through online app stores, such as Google Play, Black-Berry World, the Apple App Store, and the Windows Phone marketplace. The number of available products is amazing, with Google Play alone offering 700,000 apps at the end of 2012 (Mojica *et al.*, 2014).

Huge penetration of mobile devices, in particular, smart phones, and the development of mobile broadband are important factors for the development of Mobile applications and services (Hammershoj *et al.,* 2010). The popularity of smart phones has increased tremendously expressed by the doubling of the number of sold smart phones from 149 million units in 2010 to 297 million units in 2011. The market share of smart phones has also increased from 19% of total sold mobile phones to 31%. It is obvious that the popularity of smart phones is not due to telephony or SMS that can also be offered by both feature phones and low cost

REFERENCES

Agrawal, A. & Patel, S. (2012). An Approach to Analysis Software Reusability. *International Journal of Advanced Research in Computer Science, 3(3),* pp. 286-291.

AL-Badareen, A., Selamat, M., Jabar, M., Din, J. & Turaev, S. (2011). Reusable Software Component Life Cycle. *International Journal of Computers, 5(2),* pp. 191-199.

Aloysius, A. & Arockiam, L. (2012). Coupling Complexity Metric: A Cognitive Approach. *International Journal of Information Technology & Computer Science, 4(9),* pp. 29-35.

Amin, F., Mahmood, A. & Oxley, A. (2011). Reusability Assessment of Open Source Components for Software Product Lines. *International Journal of New Computer Architectures and their Applications (IJNCAA), 1(3),* pp. 519-533.

Babu, G. & SRIVATSA, S. (2009). ANALYSIS AND MEASURES OF SOFTWARE REUSABILITY. *International Journal of Reviews in Computing, 1(5),* pp. 41-46.

Bauer, V., Eckhardt, J., Hauptmann, B. & Klimek, M. (2014, June). An exploratory study on reuse at google. *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*. Hyderabad. ACM. pp.14-23.

Briand, L., Devanbu, P. & Melo, W. (1997). An Investigation into Coupling Measures for C++. *Proceedings of the 19th International Conference on Software Engineering*. Boston. IEEE. pp. 412-421.

Capiluppi, A., Stol, K. & Boldyreff, C. (2011). Software reuse in open source: A case study. *International Journal of Open Source Software and Processes (IJOSSP), 3(3),* pp.10-35.

Cheikhi, L., Al-Qutaish, R., Idri, A. & Sellami, A. (2014). Chidamber and Kemerer Object-Oriented Measures: Analysis of their Design from the Metrology

Perspective. *International Journal of Software Engineering & Its Applications, 8(2),* pp. 359-374.

Cramer, H., Rost, M., Belloni, N., Bentley, F. & Chincholle, D. (2010). Research in the large using app stores, markets, and other wide distribution channels in Ubicomp research. *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct.* Copenhagen. ACM. pp. 511-514.

Concas, G., Marchesi, M., Murgia, A., Pinna, S., & Tonelli, R. (2010). Assessing traditional and new metrics for object-oriented systems. *Proceedings of the ICSE Workshop on Emerging Trends in Software Metrics*. South African. ACM. pp. 24-31.

Dubey, S. & Rana, A. (2010). Assessment of usability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes, 35(6),* pp. 1-4.

Gandhi, P., Bhatia, P. K., Kumari, U. & Bhasin, S. (2011). Estimation of generic reusability for object-oriented software an empirical approach. *ACM SIGSOFT Software Engineering Notes, 36(3),* pp. 1-4.

Gandhi, P. & Bhatia, P. (2010). Reusability Metrics for Object-Oriented System: An Alternative Approach. *International Journal of Software Engineering (IJSE), 1(4),* pp. 63-72.

Garrido, A., Rossi, G. & Distante, D. (2009). Systematic improvement of web applications design. *Journal of Web Engineering, 8(4),* pp. 371-404.

Ghosheh, E., Black, S., & Qaddour, J. (2008). Design metrics for web application maintainability measurement. *Proceedings of the International Conference on Computer Systems and Applications*. Doha. IEEE. pp. 778-784.

Glasberg, D., El-Emam, K., Memo, W. & Madhavji, N. (2000). *Validating object-oriented design metrics on a commercial java application*. Canada: National Research Council.

Goel, B. & Bhatia, P. (2013). Analysis of reusability of object-oriented systems using object-oriented metrics. *ACM SIGSOFT Software Engineering Notes, 38(4),* pp. 1-5.

Gui, G. & Scott, P. (2009). Measuring Software Component Reusability by Coupling and Cohesion Metrics. *Journal of computers, 4(9),* pp. 797-805.

Hammershoj, A., Sapuppo, A. & Tadayoni, R. (2010). Challenges for mobile application development. *Proceedings of the 14th International Conference on Intelligence in Next Generation Networks (ICIN)*. Berlin. IEEE. pp. 1-8.

Hokamura, K., Ubayashi, N., Nakajima, S., & Iwai, A. (2010). Reusable aspect components for web applications. *Proceedings of the TENCON 2010-2010 IEEE Region 10 Conference*. Fukuoka. IEEE. pp. 1059-1064.

Hristov, D., Hummel, O., Huq, M. & Janjic, W. (2012). Structuring Software Reusability Metrics for Component-Based Software Development. *Proceedings of the Seventh International Conference on Software Engineering Advances*. Lisbon. IARIA. pp. 421-429.

Huang, D., He, B. & Miao, C. (2014). A Survey of Resource Management in Multi-Tier Web Applications. *IEEE Communications Surveys & Tutorials, 16 (3)*, pp. 1574-1590.

Huy, N. & Thanh, D. (2012). Developing apps for mobile phones. *Proceedings of the 7th International Conference on Computing and Convergence Technology (ICCCT)*. Seoul. IEEE. pp. 907-912.

Iqbal, N. & Qureshi, M. (2012). Improvement of Key Problems of Software Testing in Quality Assurance. *Science International Journal -Lahore,21(1)*, pp. 25-28.

Jatain, A. & Gaur, D. (2012). Estimation of component reusability by identifying quality attributes of component: a fuzzy approach. *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*. Avinashilingam University. ACM. pp. 738-742.

Kakkar, P., Sharma, M. & Sandhu, P. (2012). Modeling of Reusability of Procedure based Software Components using Naive Bayes Classifier Approach. *International Journal of Computer Applications, 55(15)*, pp. 12-17.

Karthikeyan, T. & Geetha, J. (2012). A Study and Critical Survey on Service Reusability Metrics. *International Journal of Information Technology & Computer Science, 4(5)*, pp. 25-31.

Kumar, A. (2012). Measuring Software Reusability using SVM based Classifier Approach. *International Journal of Information Technology and Knowledge Management, 5(1)*, pp. 205-209.

Kumari, U. & Bhasin, S. (2011). Application of object-oriented metrics To C++ and Java: a comparative study. *ACM SIGSOFT Software Engineering Notes, 36(2)*, pp. 1-10.

Kaur, N. & Singh, A. (2013). A Metric for Accessing Black Box Component Reusability. *International Journal of Scientific & Engineering Research, Volume 4(7)*, pp. 1114-1121.

Lionbridge (2014). *Mobile Web Apps vs. Mobile Native Apps. How to Make the Right Choice*. Retrieved on April 17, 2014, from http://www.lionbridge.com/files/2012/11/Lionbridge-WP_MobileApps2.pd.

Manhas, S., Sandhu, P., Chopra, V. & Neeru, N. (2010). Identification of Reusable Software Modules in Function Oriented Software Systems using Neural Network Based Technique. *World Academy of Science, Engineering and Technology, 4(1)*, pp. 18-28.

Mikkonen, T. & Taivalsaari, A. (2011). Apps vs. Open Web: The Battle of the Decade. *Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development*. Santa Monica. MSE. pp. 22-26.

Mohagheghi, P. & Conradi, R. (2008). An empirical investigation of software reuse benefits in a large telecom product. *ACM Transactions on Software Engineering and Methodology (TOSEM), 17(3)*, pp. 13-21.

Mojica Ruiz, I., Adams, B., Nagappan, M., Dienst, S., Berger, T., & Hassan, A. (2014). A large scale empirical study on software reuse in mobile apps. *Software IEEE, 31(2)*, pp. 78-86.

Monga, C., Jatain, A. & Gaur, D. (2014). Impact of quality attributes on software reusability and metrics to assess these attributes. *Proceedings of the Advance Computing Conference (IACC).* Gurgaon. IEEE. pp.1430-1434.

Narasimhan, V., Parthasarathy, P. & Das, M. (2009). Evaluation of a suite of metrics for component based software engineering (CBSE). *Issues in Informing Science and Information Technology, 6(5/6)*, pp. 731-740.

Nuruzzaman, M., Hussain, A. & Tahir, H. (2013). Towards Increasing Web Application Development Productivity through Object-Oriented Framework. *International Journal of Future Computer and Communication, 2(3)*, pp.220-225.

Okike, E. (2010). A Pedagogical Evaluation and Discussion about the Lack of Cohesion in Method (LCOM) Metric Using Field Experiment. *International Journal of Computer Science Issues (IJCSI), 7(2)*, pp. 36-43.

Patwa, S. & Malviya, A. (2012). Reusability metrics and effect of reusability on testing of object oriented systems. *ACM SIGSOFT Software Engineering Notes, 37(5)*, pp. 1-4.

Pylkki, V. (2013). *Evaluating application generators for multi-platform mobile application development*. University of Tamper: Master's Thesis.

Redin, R., Oliveira, M., Brisolara, L., Mattos, J., Lamb, L., Wagner, F. & Carro, L. (2008). *On the use of software quality metrics to improve physical properties of embedded systems*. Brazil: Institute of Informatics (UFRGS).

Sridhar, M., Srinivas, Y. & Krishna Prasad, M. (2013). Software reuse in a paralysis dataset based on categorical clustering and the Pearson distribution. *Journal of King Saud University-Computer and Information Sciences, 26(3)*, pp. 347-354.

Sagar, S., Nerurkar, N. & Sharma, A. (2010). A soft computing based approach to estimate reusability of software components. *ACM SIGSOFT Software Engineering Notes, 35(5)*, pp. 1-5.

Sagayaraj, S. & Ganapathy, G. (2011). Extraction of method signatures from ontology towards reusability for the given system requirement specification. *Proceedings of the International Conference of Applied and Engineering Mathematics*. London. WCE. pp. 988-994.

Sandhu, P., Kakkar, P. and Sharma, S. (2010). Mechanical and Electrical Technology. *Proceedings of the 2nd International Conference on International Conference On Mechanical Engineering And Technology (ICMET)*. Singapore. IEEE. pp. 769-773.

Serrano, N., Hernantes, J. & Gallardo, G. (2013). Mobile Web Apps. *Software, IEEE, 30(5)*, pp. 22-27.

Sharma, A., Grover, P. S. & Kumar, R. (2009). Reusability assessment for software components. *ACM SIGSOFT Software Engineering Notes, 34(2)*, pp.1-6.

Singaravel, G., Palanisamy, V. & Krishnan, A. (2010). Overview analysis of reusability metrics in software development for risk reduction. *Proceedings of International Conference on Innovative Computing Technologies (ICICT)*. Tamil Nadu. IEEE. pp. 1-5.

Singh, S., Thapa, M., Singh, S. & Singh, G. (2010). Software Engineering- Survey of Reusability Based on Software Component. *International Journal of Computer Applications, 8(12)*, pp. 39-42.

Singh, Y., Bhatia, P. K. & Sangwan, O. (2011). Software reusability assessment using soft computing techniques. *ACM SIGSOFT Software Engineering Notes, 36(1)*, pp. 1-7.

Suri , P. & Garg, N. (2009). Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse. *International Journal of Computer Science and Network Security, 9(5)*, pp. 237-248.

Taibi, F (2014). Empirical Analysis of the Reusability of Object-Oriented Program Code in Open-Source Software. *International Journal of Computer, Information Science and Engineering, 8 (1)*, pp. 4553-4557.

Taibi, F. (2013). Reusability of open-source program code: a conceptual model and empirical investigation. *ACM SIGSOFT Software Engineering Notes, 38(4)*, pp. 1-5.

Trivedi, P. & Kumar, R. (2012). Software Metrics to Estimate Software Quality using Software Component Reusability. *International Journal of Computer Science Issues (IJCSI), 9(2)*, pp. 144-149.