OOCQM: Framework for calculating object-oriented code quality using quality factors and code metrics

Author
Asma Shaheen
00000171320
MS-16(CSE)

Supervisor
Dr. Usman Qamar

Department of Computer Engineering
College of Electrical and Mechanical Engineering
National University of Sciences and Technology
Islamabad
October 2019

In the name of Allah most beneficent most merciful

أَلَمْ تَرَ أَنَّ ٱللَّهَ يَعْلَمُ مَا فِى ٱلسَّمَٰوَٰتِ وَمَا فِى ٱلْأَرْضِ ۖ مَا يَكُونُ مِن
نَّجْوَىٰ ثَلَٰثَةٍ إِلَّا هُوَ رَابِعُهُمْ وَلَا خَمْسَةٍ إِلَّا هُوَ سَادِسُهُمْ وَلَآ أَدْنَىٰ
مِن ذَٰلِكَ وَلَآ أَكْثَرَ إِلَّا هُوَ مَعَهُمْ أَيْنَ مَا كَانُوا۟ ۖ ثُمَّ يُنَبِّئُهُم بِمَا عَمِلُوا۟ يَوْمَ
ٱلْقِيَٰمَةِ ۚ إِنَّ ٱللَّهَ بِكُلِّ شَىْءٍ عَلِيمٌ ﴿٧﴾

Have you not considered that Allah knows what is in the heavens and what is on the earth? There is in no private conversation three but that He is the fourth of them, nor are there five but that He is the sixth of them - and no less than that and no more except that He is with them [in knowledge] wherever they are. Then He will inform them of what they did, on the Day of Resurrection. Indeed Allah is, of all things, Knowing. [58:7]

OOCQM: Framework for calculating object-oriented code quality using quality factors and code metrics

Author
ASMA SHAHEEN
00000171320

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Computer Software Engineering

Thesis Supervisor:
DR. USMAN QAMAR
Thesis Supervisor's
Signature:_____

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
OCTOBER 2019

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student
Asma Shaheen
Registration Number
00000171320

Signature of Supervisor
Dr. Usman Qamar

Copyright Statement

This page is intentionally left blank

ACKNOWLEDGEMENTS

"This is by the Grace of my Lord to test me whether I am grateful or ungrateful! And whoever is grateful, truly, his gratitude is for (the good of) his own self, and whoever is ungrateful, (he is ungrateful only for the loss of his own self). Certainly! My Lord is Rich (Free of all wants), Bountiful" [An-Naml: 40]

I am thankful to NUST College of Electrical and Mechanical Engineering, for providing me an opportunity for Masters Research. First of all I am thankful to my thesis supervisor, Dr. Usman Qamar, who guided me throughout my research, with his valuable knowledge. I attribute the level of my Master's degree to his encouragement and effort and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier supervisor. I would also like to thank my family for the support they provided me through my entire life.

Dedicated
to
To my parents and siblings

ABSTRACT

Purpose: *Source code is core of Software engineering. Source code of good quality can be maintained and upgraded easily. Measurement of code quality is important from various views like from developer's point of view, from manager's point of view and from end user's point of view. Main issue in measurement of quality is that quality it needs to be measured from different aspects of software.* ***Methodology:*** *Source code quality can be broken down in factors like maintainability, reusability, change proneness, testability and stability. These quality factors are high level representation of code quality and cannot be measured directly. In order to calculate quality factors there is need for calculations for certain aspects of software code at lower level. Research in field of source code quality has proposed enormous amount of code metrics that measure quality in different aspects like complexity, size, coupling, cohesion and abstractness etc. This study proposes a framework named Object Oriented Code Quality Meter (OOCQM) for measuring source code quality of object-oriented code using low level code metrics and high-level quality factors. Code metrics has a proven relationship with quality factors. This relationship is used to calculate a numerical value for quality factors based on metric values. It is observed that all selected metrics has negative correlation with mapped quality factor. Quality factors scores are aggregated and used to depict quality of code in numerical form. A PHP based tool is developed to validate the results.* ***Results:*** *Framework results are compared with Maintainability Index (MI) which is popular quality measure in terms of code maintainability. OOCQM measures code quality correctly as quality results are correct according to MI. This framework provides more detail at individual quality factors level. OOCQM is compared with few other tools developed for quality measurement. Comparison shows that this tool supports more quality factors analysis than other tools.*

Keywords: *OO Code Quality, Code Metrics, Software Quality Factors, Source Code Quality Calculation*

Table of Contents

List of Figures

List of Tables

List of Equations

CHAPTER 1

INTRODUCTION

"Quality is not an act; it is a habit" *Mark E. Daggett*

Software development is quiet mature industry which is flourishing rapidly in many directions. Software is core of modern technology advancements. New technologies like Artificial Intelligence, Machine Learning, and Data Mining etc. are dependent on software.

This time is called digital era as digitization is involved in every field of life. Need of digitization has increased use of software exponentially. More and more software are being developed by large to small software companies and even individual programmers to cope with the demand of software development. There are many types of software like system software, application software, embedded software, web based and desktop software etc. Software can also be classified based on language in which they are developed.

Software development is a complex and dynamic process which requires a team effort with diverse skills as software development starts from a problem statement and reaches to a state of full fledge deliverable product. Development process includes requirement engineering (requirement gathering and specification), analysis, design, coding, testing and deployment. Quality of a software is a key consideration at every stage of software development. A compromise on quality not only reduces trust on software and on company that delivered software. It is complex to enhance and maintain a low-quality

software.

Measuring quality of a software is challenging due to several reasons and few of these reasons are:

- Diversity in software types
- Lot of programming languages
- Lack of standards for quality measures
- Software industry focuses mainly on external quality of software
- No standard tools available for measuring software quality

Software quality factors are used as base for measuring quality. A quality factor is a set of nonfunctional requirements or quality attributes that are well define and can be grouped in a set.

McCall derived 11 quality factors related to software products. These quality factors can be divided in three groups, factors related to product revision, operation and transition. Correctness, usability, reliability, efficiency and integrity fall in group of factors that are related to software operation. Software transition quality factors are portability, re usability and interoperability. Maintainability, flexibility and testability are software revision related quality factors. [1]

Another interesting thing about software quality is that is a multifaceted concept. Software quality can be viewed from five perspectives. These five views include transcendental, user, manufacturing, product and user-based view. Measurement of quality depends on definition of quality from selected perspective of quality [2]

Different quality models have been proposed by researchers working in field of software quality to describe relation between different software characteristics. Models are based on quality factors. Popular models of quality are McCall's quality model and SO 9126. Software quality models can be divided in types like quality definition models, quality assessment models and quality prediction models. All three types of

quality models are criticized for not being widely applicable because requirements for their application are not briefly described. [3]

Software quality metrics are used to measure quality of a software. A software quality metric is measurable criteria of a software that can indicate goodness or badness of a software. Software quality literature provides lot of metrics for measuring different aspects or factors of software. Metric is a lower level quality measure. A criterion for good metric is that It should have clear objective, it should be valid and useful, repeatable and comparable. Software metrics history starts from size and structure metrics. Famous metrics suits from history are Halstead metrics that calculates effort, difficulty and volume of software, CK metrics suit consists on metrics related to object-oriented features. Cyclomatic complexity is a popular metric that is applicable to all types of software code.

There are lot of programming languages and measuring quality of code is somewhat language specific especially code parsing part is different for every language due to difference in syntax and semantic rules. PHP is an open source web development language that is been used since last thirty years. PHP was developed in 1994 by Rasmus Lerdorf. It was developed in C. PHP has evolved a lot and has become a mature and popular web development language that is easy to learn and efficient to accomplish web application requirements.

1.1 Background and Motivation

Measurement leads to standardization which makes comparisons easy and reliable in all fields of life. We cannot compare two things adequately without standard measures. Same is the case for software code quality. We need standard measures to evaluate code quality. Metrics are considered somewhat standard measure for analyzing software code quality. With reference to IEEE Software Engineering Standards “Metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute”.

There are four popular programming paradigms Procedural Programming (PP), Object Oriented Programming (OOP), Aspect Oriented Programming (AOP) and Feature Oriented Programing (FOP). PP is start of programing and still used. OOP got popular as size of code and complexity of software increased. OOP provided lot of good features and made coding easy for large and complex software. AOP and FOP are not as popular as OOP and are used for specific scenarios and languages. Measuring quality of each type of software code is somewhat different form other type. As OOP is most popular programming paradigm same is the case with OOP metrics.

There is a great need for standardization of all these metrics so that choosing suitable set of metrics for a certain type of code is easy and results of selected metrics can depict actual quality of in better way. Lot of code quality metrics are proposed in literature but application of those metrics in industry is quite difficult and not encouraged by software industry experts. So, there is a big gap between quality measures proposed in literature and their actual usage in industry.

1.2 Objective and Contribution

Importance of measurement is described by a famous quote by Tom DeMarco "*You can't control what you can't measure".* In order to measure the quality of software code many frameworks are proposed that measure different aspects of code quality. These frameworks are designed mostly for specific programing languages and measure quality for specific scenarios. Quality measurement frameworks are based on different quality metrics. Code quality metrics are standardized measurements of code that work equally good for all programming languages and code structures.

More than 300 metrics have been proposed for measurement of source code quality by different researchers. Most of the proposed metrics are suggested and implemented on Object Oriented Programming code. for Procedural Programming (PP) code metrics, Aspect Oriented Programming (AOP) code metrics and Feature oriented Programing (FOP) code metrics are present in literature but those metrics are less in number and low in popularity as compared to object-oriented code metrics [4]

There is great need to analyze proposed metrics and choose best set or suit of metrics for code of a software project. Another issue in analyzing a quality metric value is the criteria for good or bad software based on metric result. In fact, defining criteria for metric value is critical as this criteria or threshold varies based on software type, size and other conditions. Therefor deciding relative thresholds for metrics is important. This will help in better interpretation of metrics results.

This study is focused on measuring quality of software projects using metrics. Therefore, metrics are analyzed in detail including their calculation, validity, importance etc.

A web-based application is developed as a tool according to this study that can analyze code quality of any PHP based program or project and can show quality results in a manner that can facilitate developer/managers to understand insight of code and take better decisions about project.

1.3 Outline

This research work is written in following sequence that is quite traditional in this field, chapter 2 provides the detail description of previous research work done in field of software code quality including quality metrics, quality models and quality frameworks and their significance. Chapter 3 describes the methodology used to design the framework for picking suitable metrics for code. In chapter 4 structure of tool used to measure quality is described. Chapter 5 focuses on validation of proposed framework. In chapter 6 framework results are validated. Chapter 7 summarizes the whole research work and future research possibilities are described

1.4 Summary

software metrics are basic measure for software quality at lower level. But taking managerial decisions based on metrics results is not possible. Also, there are lot of software code metrics that are proposed by researchers for analyzing software code quality. This research work is

focused on designing a software code quality framework that is based on quality metrics and can provide software code quality information at higher level which can help in managerial decisions. All types of software code quality metrics are analyzed and suitable metrics are selected for this framework.

There are some code quality analyzing tools for different languages like Java and C etc. There is no good tool that can analyze code quality of PHP language according to metrics proposed in research literature

CHAPTER 2

LITERATURE REVIEW

Software code is actual implementation or realization of software requirements and it controls operations of designed system. Software code quality metric is a measure that defines any aspect or characteristic of software behavior. Code quality metrics have proven correlation with software characteristics. Metrics provide information to developers about internal characteristics of software that can affect software quality. Many metrics are also beneficial for managers and end users.[5]

This chapter will highlight previous work on software code quality. This chapter is divided in four main sections, software quality characteristics, software quality models, code quality metrics and code quality analysis frameworks/tools used for code quality measurement

2.1 Software Quality Characteristics/Factors

Software quality can be broken down into several characteristics/attributes. In 1977 a study was conducted by McCall, Richards and Walters to identify the factors that can be used to define quality of a software. In this study first quality factors gathered from literature and named as candidate factors. A set of rules devised to choose factors from list of all factors picked from literature. This set of rules included following rules:

- A characteristic that shows contribution in quality of software
- A characteristic that shows how it is related to user

- Relation to cost
- Relation to software product

55 candidate factors were identified from literature based on above rules. But it was noted that there was redundancy in list of these 55 factors and also it was not manageable to analyze quality with such long list of factors. So, a set of guidelines developed to choose prime factors from these candidate factors. Those guidelines are listed below:

- Only pick user oriented and software-oriented factors
- Group synonyms together
- Group terms which are logically related
- Reduce factors to a manageable number

After grouping the factors based on above criteria following factors were finalized as candidate for quality measuring factors:

- Correctness
- Portability
- Reliability
- Maintainability
- Reusability
- Efficiency
- Flexibility
- Interoperability
- Integrity
- Usability
- Testability[6]

2.1.1 Correctness

Correctness is checked by traceability, completeness and consistency. For a software/program it can be defined as how much the software/program fulfills its specifications and how much it can fulfill user's objectives. Sub factors of correctness are acceptability, completeness, consistency, expression, performance and validity [6] This factor is included in quality attributes or factors by various models like McCall model , Boehm model and ISO/IEC 9126[7]

2.1.2 Portability

Portability is collection of compatibility and transferability. It can be defined as amount of effort required to shift a program from one environment or hardware to another.
Portability is considered as a required element for the majority of software. In this time of digitization and technology where software is used in all type of day to day tasks and businesses with different technologies by different vendors, software products need to be used in different environments. Therefore, support for multiple platforms or environments is key requirement for software products. Software product of any category or scale should have ability to easily migrate to newer, different and better environments.[8]

2.1.3 Reliability

This attribute refers to availability, accuracy, robustness, tolerance and precision of a software program. It can be defined as expected precision of desired output/results/performance. Error tolerance, consistency, simplicity and accuracy are criteria to measure reliability of a software.
Another definition of reliability is the probability that a software will execute without any issue for a specified period in conditions that are previously specified. [9]

2.1.4 Maintainability

It refers to ease with which an issue in software can be fixed or software can be upgraded to add new features or improve any existing feature. [9]

Or it can be amount of time and effort required to fix an issue in software. Maintainability covers following points:

- Fixing bugs in existing software
- Enhancing software by improving any existing feature
- Expanding software to add new features

2.1.4 Reusability

This attribute indicates how general a piece of software is that it can be re used for other software applications. This attribute belongs to software internal quality[10]

2.1.5 Efficiency

This is attribute belongs to software internal quality. It covers resource and time utilization

It can be defined as capability of a software to deliver specified performance with specified resources within specified time[10]

2.1.6 Flexibility

Flexibility refers to ease with which a program can be changed and can be added to other programs. Modularity, simplicity, self-descriptiveness and generality are sub factors of flexibility [11]

2.1.7 Interoperability

Interoperability is the capability of two or more components of software to cooperate

regardless of differences in interface, language, and platform of execution. It can be described as a scalability of reusability[12]

It refers to effort required to join one software with other. Interoperability requires use of standard protocols so software can be deployed easily with different type of software.

2.1.8 Integrity

It refers to protection against un authorized access of the software. It also refers to maintain accurate state of data in shared environment. Sub attributes of integrity are security and privacy.

Integrity of a software is tested by following points:

- Software is complete
- Software is protected
- Software performs all intended functionality[13]

2.1.8 Usability

This attribute refers to effort needed to operate, learn, prepare input and understand results of a program. Usability is external quality attribute. Usability is tested in operation and training of software. Communicativeness is also a test criterion of usability. This attribute cannot be measured only from code.

Usability means to how the software system communicates with the user. Usability has following five attributes:

- Learnability
- Efficiency
- User retention over time
- Error rate
- satisfaction[14]

2.1.9 Testability

In literature sometimes it is also shown as sub attribute of maintainability. It is internal quality attribute. It refers to the effort needed to test a software to make sure that intended functionality is provided.

Testability can be checked by traceability, structuredness and legibility[15]

2.2 Software Quality Models

Quality model can be defines as a model which is developed to describe quality, analyze quality or predict quality.[3]

This section will provide a brief review of popular software quality models proposed in literature. Popular quality models are:

- ISO 9126 model
- McCall model
- Boehm's model
- FURPS model
- Dromey's model
- WBA quality model

2.2.1 ISO 9126

Software quality can be broken down into three views. Internal quality of software, external quality of software and quality in use. Internal quality refers to characteristics of software that can be measured from code without executing code. External quality refers to properties of software that are examined or measured during execution. Quality in use refers to concerns of users during operation of software. Internal quality affects external quality of software which in turn affects quality in use. ISO 9126 divides product quality into following 6 characteristics:

- Functionality
- Usability
- Reliability
- Efficiency
- Portability
- Maintainability

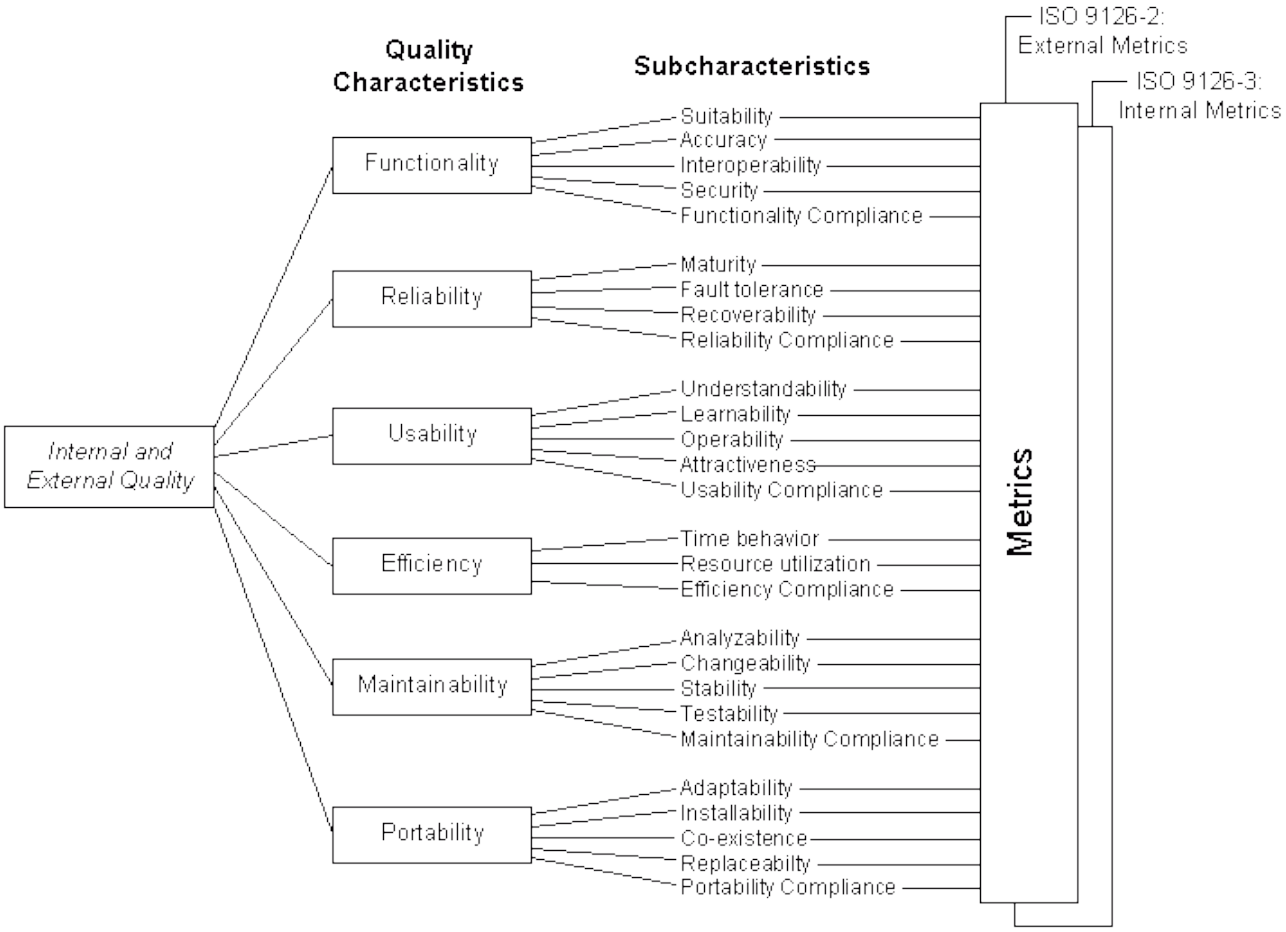

Figure 2.1 ISO 9126 [16]

These characteristics are further subdivided in 27 sub characteristics. [17]

This division is shown in following table 2.1

Table 2.1 ISO 9126 Quality Factors and Sub factors

Quality Factor	**Sub Factors**
Functionality	Suitability Accuracy Security Interoperability
Usability	Understandability Learnability Operability Attractiveness
Reliability	Maturity Fault tolerance Recoverability
Efficiency	Time Utilization Resource Utilization
Maintainability	Analyzability

	Changeability Stability Testability
Portability	Adaptability Install ability Co-existence Replaceability

2.2.2 McCall Model

This model was developed to analyze the relationship between quality criteria of software and external factors[18]

This model proposes quantification of quality through a hierarchical approach. Top layer consists of factors, these factors are sub divided in criteria and criteria is measured via metrics. This model proposes that quality belongs to three stages of software product. Product operation, product revision and product transition[2]

Quality characteristics that belong to use of product are collected under product operation category. Quality attributes that belong to maintenance and environment of software product are gathered under product revision category. Product transition bundles the quality characteristics that belongs to shifting a software product to a new environment. Following quality factors are identified in this model:

- Correctness
- Reliability
- Efficiency

- Integrity
- Usability
- Maintainability
- Testability
- Flexibility
- Portability
- Reusability
- Interoperability

These quality factors are mapped with a stage of software product. Usability, integrity, efficiency, reliability and correctness are measured in operation of product by users or trainers of product. Maintainability, testability and flexibility are important revision of product when a new version of product is created based on new features required or for fixes of any identified issues. Portability, reusability and interoperability are important when product is moved from one environment to another a bit different environment These quality factors are further sub divided in quality criteria and those criteria are measured via metrics. Listing of factors and quality criteria is shown in figure 2.2

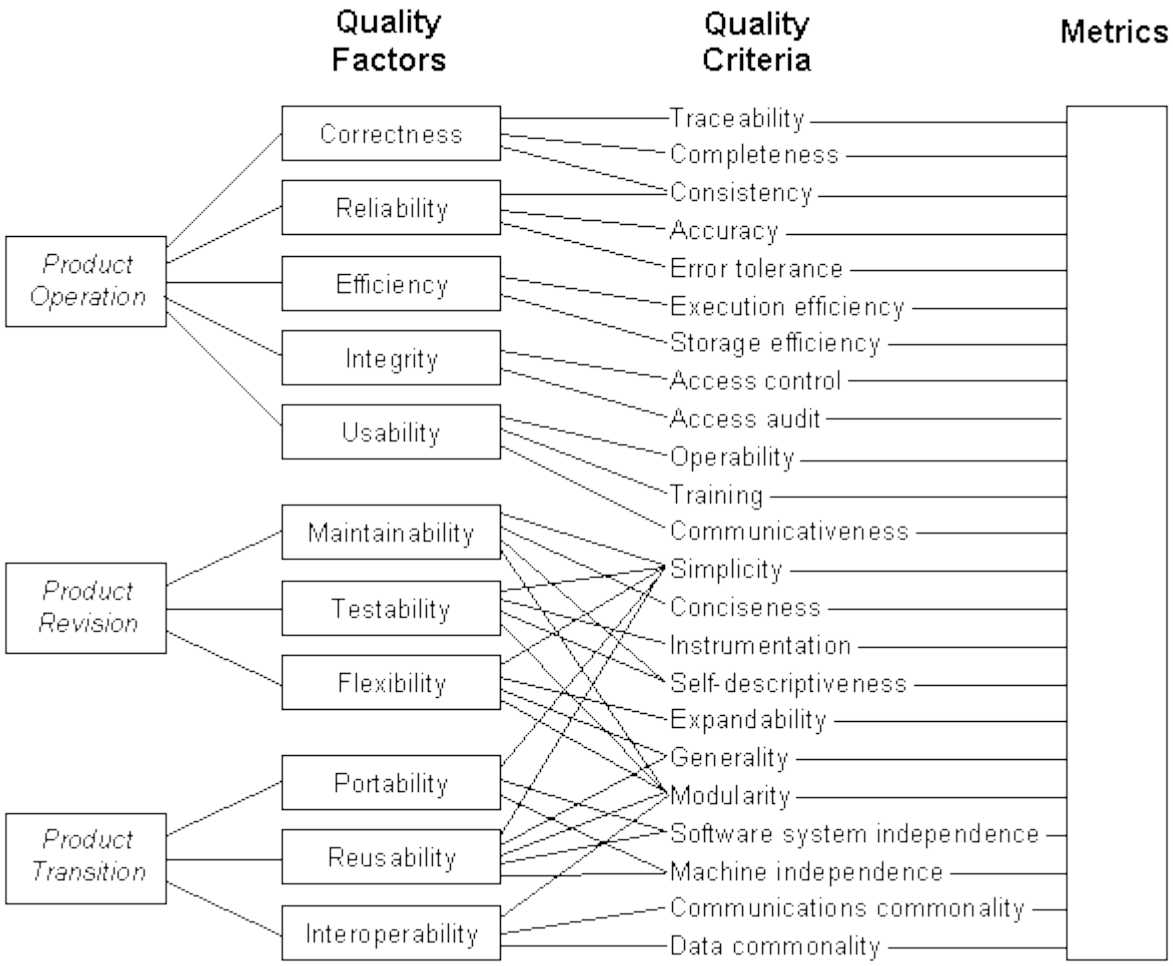

Figure 2.2 McCall Model [19]

2.2.3 Boehm's Model

This model was proposed by Boehm to represent software quality in a quantitative form through automatic calculation. It consists of three levels of software characteristics[20] Overall quality of software also called general utility is top level in hierarchy of characteristics.

This general utility or overall quality s divided in three factors portability, usability also called utility and maintainability. Usability is further broken down into reliability, efficiency and human engineering. Maintainability is further divided in testability, modifiability and understandability.

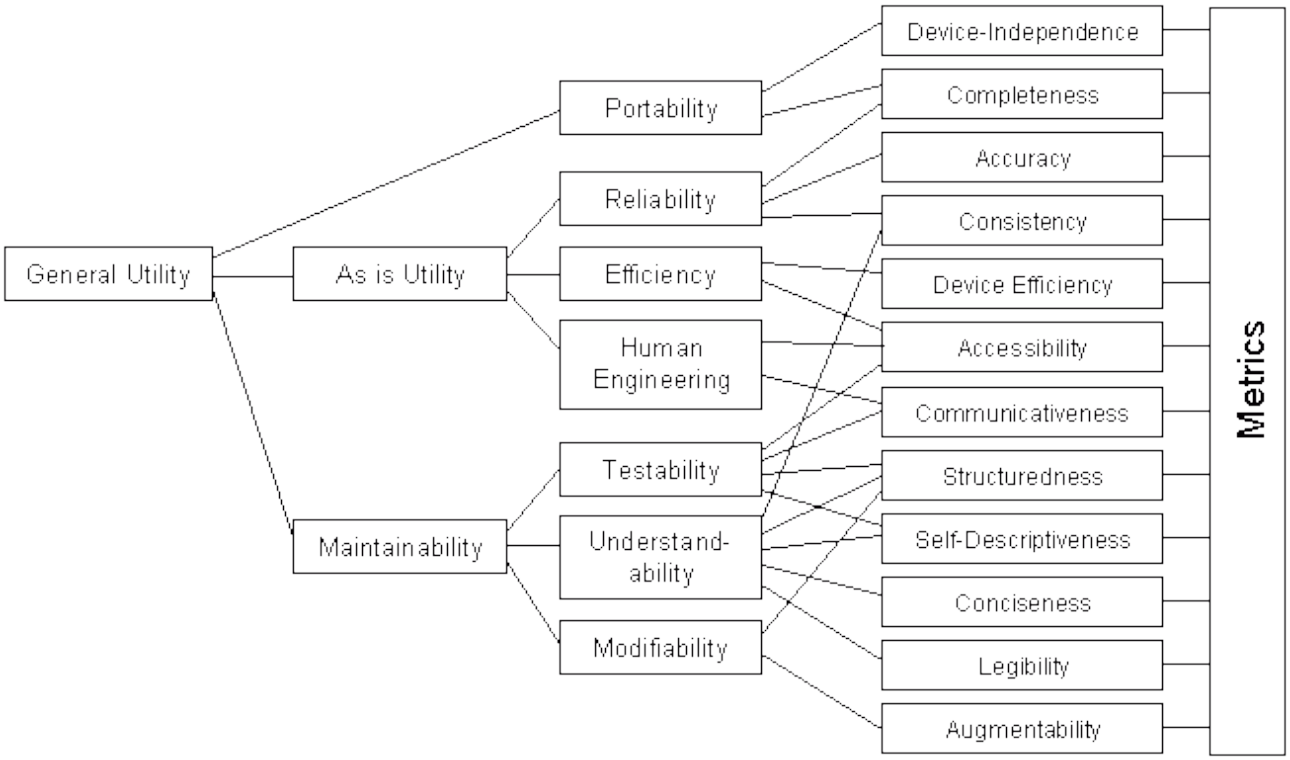

Figure 2.3 Boehm's Model [21]

2.2.4 FURPS Model

This model divides software quality attributes in two types functional and nonfunctional. Nonfunctional requirements (the URPS) include following characteristics:

- Usability
- Reliability
- Performance
- Supportability

Functionality includes list of features, capabilities of software and also security of software.

Usability refers to aesthetics, human factors and user interface consistency. It also includes help, training material and documentation of software.

Reliability refers to frequency of failure and severity of failure, predictability,

recoverability, mean time between failures (MTBF) and accuracy. Performance refers to functional requirements of the software like efficiency, speed, availability, response time, accuracy, throughput, resource usage and recovery time. Supportability includes extensibility, testability, maintainability, adaptability, configurability, compatibility, install ability, serviceability, and localizability.[22]

Figure 2.4 shows structure of FURPS model

Figure 2.4 FURPS Model [22]

2.2.5 Dromey's Software Quality Model

This model was proposed to analyze quality of software product in three phases:

- Requirement elicitation
- Design
- Implementation

The main focus of this model is on implementation quality measures. Implementation quality measures are divided in two levels[23]

First level contains:

- Correctness
- Internal quality measures

- Contextual measures
- Descriptive measures

Correctness is sub divided in correctness of functionality and reliability of software. Internal measures check maintainability, efficiency and reliability of software. Contextual quality measures include maintainability, reusability, reliability and portability. Maintainability, usability, reliability and efficiency fall in category of descriptive measures. Figure 2.5 outlines the structure of this model.

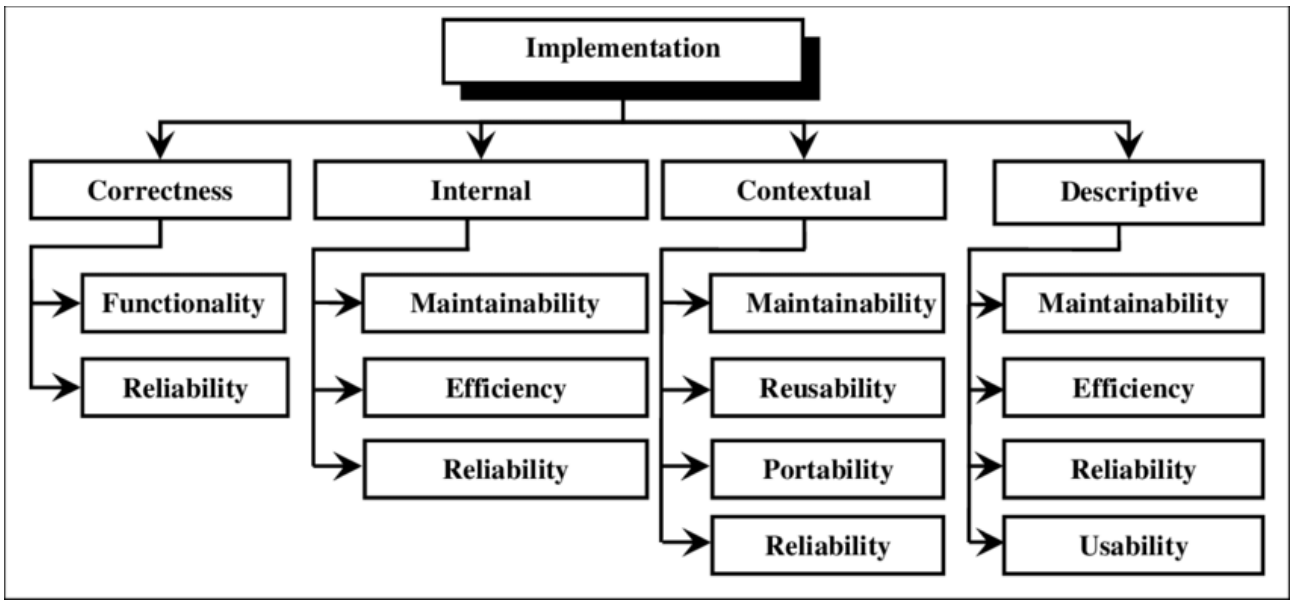

Figure 2.5 Dromey's Model [24]

2.2.6 WBA Quality Model (WBAQM)

Analyzing quality of web-based software is quite complex because it depends on multiple criteria. and multiple attributes. [25]

Web Based Application (WBA) has three perspectives for quality measurement:

- Developers perspective
- Owner perspective
- Visitor perspective

This model focuses on developers and visitors concerns. Following quality factors are identified from developers concerns:

- Maintainability
- Portability
- Reusability

Quality factors identified from visitor perspective are:

- Usability
- Accessibility
- Content
- Functionality
- Security

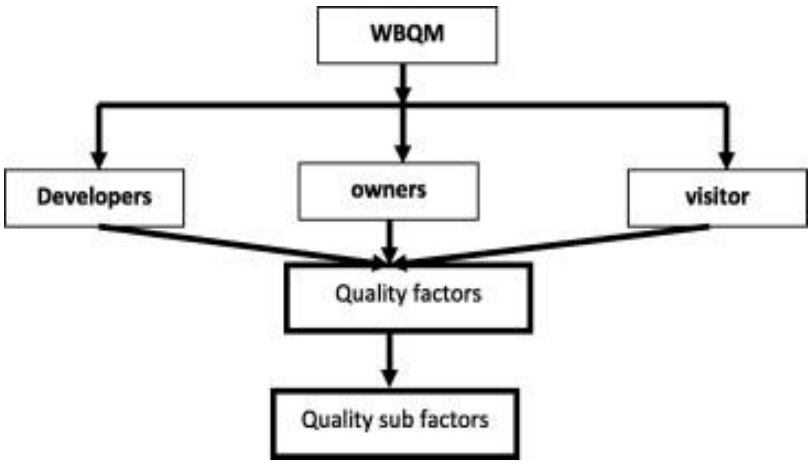

Figure 2.6 WBQA Quality Model [26]

2.3 Software Code quality Metrics

Code quality metrics are measures that evaluate some aspect of software code. In literature there are hundreds of code metrics proposed based on software code type and measurement requirements. This section will present software code metrics that are related to object-oriented code and are popular in literature.

2.3.1 SLOC Metric

This metric is considered most basic and simple metric that can be helpful in analyzing code quality. This metric is also used in calculating some other metrics. This metric was initially proposed for machine language or assembly language. SLOC in higher level languages is used as predictor for other metrics. For example, SLOC can be used for predicting defect density as it has negative correlation with code size. SLOC is also used as covariate to determine development effort used to create code artifact as well as the effort that will be required to maintain the software. SLOC metric was proposed for procedural programs. [27]

2.3.2 Halstead's Metrics

Halstead's metrics suit is one of oldest metrics suite used to analyze complexity of a software program. According to Halstead a software program is collection of tokens and these tokens fall in two categories operators and operands. Metrics proposed by Halstead are based on counting these two types of tokens.

Table 2.2 Operators used in Halstead's metrics

Operator	**Definition**
n_1	Number of operators that are distinct
n_2	Number of operands that are distinct
N_1	Total operators used in code
N_2	Total number of operands used in code
n_1*	Number of potential operators or minimum possible operators
n_2*	Number of potential operands or minimum possible operands

Halstead used operators mentioned proposed ten metrics which are listed below in Table 2. operators used in these metrics are explained in Table 2.

Table 2.2 Halstead Metrics

Metric Name	**Calculation Formula**
length(N)	$N_1 + N_2$
vocabulary(n)	$n_1 + n_2$
volume(V)	$N*\log_2 n$
Potential volume(V*)	$(2 + n_2*) \log_2 (2 + n_2*)$
Level(L)	V*/V
Difficulty(D)	1/L
Level Estimator (L^)	$2/n_1 * n_2*N_2$
Intelligent Content(I)	L^ * V
Program Effort(E)	V / L => $n_1N_2N\log_2 n / 2n_2$
Program time (PT)	E / S (S= Stroud number is count of elementary decisions per second, In software science it is set to 18)

Some Halstead's metrics can be used as quality indicators are for example difficulty(D), Effort(E) and Program Time (PT) [28]

2.3.3 Cyclometic Complexity

Complexity of a software is also used as quality indicator. McCabe used graph theory to represent complexity of source code. According to graph theory code size does not affects its complexity. It means adding new functional statements or removing some functional statements will not change complexity of program. Complexity only depends

on decision statements. More decision statements mean more paths in graph which leads to increasing complexity. For calculation of this metric a program is represented as graph in which graph node show processing of a task and an edge between two nodes shows control flow. McCabe proposed following formula to calculate Cyclometic complexity of a procedural program:

Equation 1 Cyclometic Complexity

$$V(G) = E - N + 2$$

In above equation E represents number of edges in graph and N shows count of nodes in graph. [29]

2.3.4 MOOSE/CK Metrics

This metric suit was proposed by Chidamber and Kemerer in 1991. It was first effort for analyzing object-oriented code specifically. There are six metrics in CK metrics suit. Metrics proposed in this suite are most popular in analysis of quality of object-oriented code. Most researchers who worked on code quality or bug prediction used this metric suit as base or benchmark

Weighted Methods per Class (WMC)

Calculating Cyclometic complexity for class is different from procedural programming. CK metric suite provided a new metric for calculating Cyclometic complexity of a class. WMC can be calculated by adding up complexity values of all local methods of a class. This formula is given in CK Metrics table. WMC value can range from 0 to N where N is any positive integer. WMC is measure of complexity of an object and its value can be used to assess how much time and effort is required to develop and maintain this object. Larger value of WMC indicates that methods in class are not generic and are more related to specific application. This indicates that this class has low reusability

Depth of Inheritance Tree (DIT)

DIT is the level of a class in hierarchy of inheritance. Root class has DIT value of 0. DIT value can range from 0 to N where N is any positive integer. The higher DIT means more functionality class has as it inherits all functionality and properties of super classes. Higher DIT indicates difficulties in maintenance of class. Higher DIT is also symbol of high complexity. Another concern with high DIT is violation of encapsulation as child class can access properties of parent classes.

Number of Children (NOC)

NOC indicates how many classes are directly inherited by a certain class. High NOC means more importance of class in application architecture. Generally speaking, depth is better in hierarchy instead of breath. It means NOC value should not be high specially when class is lower in inheritance hierarchy. If a class has more children then it requires more testing of all methods in class as it has to serve more scenarios.

Coupling between Objects (CBO)

Coupling refers to access of other class's methods or instances other than inheritance. It refers to access of one class's methods or objects by another class's functions. High CBO means design is not modular and rule of re usability via inheritance is violated. High CBO also make application difficult to test and modify, it also decreases re usability. CBO value indicates how easy or difficult is to maintain and test that class. CBO is measure of communication between objects

Response for Class (RFC)

RFC denotes to count of numbers of elements that can provide response to an object of class. This set of response elements is collection of all local methods of class plus all methods that are called by local methods of class. lower RFC values are ideal in terms

of low complexity, less testing, easy debugging and easy to understand the code. This metric is related to object attributes and object communication

Lack of Cohesion in Methods (LCOM)

Cohesion means how well methods of a class are using local variables/ properties of that class. High cohesion is a good for application maintenance and reusability. LCOM is calculated by counting disjoint sets of methods where one set is made by combining all methods that share a common variable. High LCOM indicates that class should be divided to more classes. This metric is related to attributes of objects[30]

Table 2.3 CK Metrics

#	**Metric**	**Calculation Formula**
1	WMC	$\sum_{i=1}^{n} Ci$
2	DIT	Number of classes between root and selected class in inheritance hierarchy
3	NOC	Number of direct childes/sub-classes
4	CBO	Number of classes in set of unique classes that access a class C or accessed by class C
5	RFC	Sum of number of methods in class and number of methods called by class methods
6	LCOM	count of sets of methods that are disjoint

2.3.5 MOOD Metrics

This metrics set also consists of 6 metrics. Metrics in this suit are presented to measure quality of code in terms of OOP features. These metrics measure polymorphism, coupling, encapsulation and inheritance.

Method Hiding Factor (MHF)

This metric measures the level of encapsulation of a class. It uses the count of class's public methods and count of class's total methods to derive a value that shows the level of visibility of class's definition. Formula is given in table below. TC denotes total classes in formula.

Attribute Hiding Factor (AHF)

AHF is used in conjunction with MHF to measure encapsulation level of a class. It uses class's public properties and total properties to calculate value.

Method Inheritance Factor (MIF)

This metric is used to measure inheritance level of a class. MIF is ratio of methods that are inherited from parent classes to total methods in a class. This is a class level metric. Formula of this metric is given in table below. TC denotes total classes in program or application. $M_i(C_i)$ are methods which are derived from parent classes and are not extended in class C_i. $M_d(C_i)$ represents methods declared in class. Higher value of MHF indicates high level of inheritance or more inherited methods in class.

Attribute Inheritance Factor (AIF)

AIF describes level of inheritance according to attributes of classes. AIF and MIF are accepted as **valid** measure of inheritance level of an application.

Coupling Factor (CF)

Coupling factor metric was proposed to measure coupling among classes. CF excludes coupling due to inheritance. CF indicates relationship size among classes in a system. is-client (C_i, C_j) is 1 if class C_i refers any object of class C_j or calls any method of class C_j. A high CF value indicates high level of coupling which in turn is an indicator of high complexity, low re usability, difficulty in understanding and difficulty in maintaining the application code. Thus, it can be said that CF measures coupling directly and other features like re usability, maintainability etc. indirectly.

Polymorphism Factor (PF)

PF measures level or potential of polymorphism in system. PF formula is given in table below. $M_o(C_i)$ refers to overridden methods. $M_n(C_i)$ denotes new methods of class C_i whereas $DC(C_i)$ refers to count of classes derived from class C_i. PF can also be used to measure amount of dynamic binding in application[31]

Table 2.4 MOOD Metrics

#	**Metric**	**Formula**
1	MHF	$\frac{\sum_{i=1}^{n}(C_m)}{n}$
2	AHF	Same as MHF methods are replaced with attributes in formula
3	MIF	$\frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_d(C_i) + M_i(C_i)}$
4	AIF	Same as MIF methods are replaced with attributes in formula
5	CF	$\frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC} is-client(C_i, C_j)]}{Tc^2 - TC}$
6	PF	$\frac{\sum_{i=1}^{n}(C_m)}{n}$

2.3.6 QMOOD

QMOOD model was proposed for analyzing high level attributes related to quality like re usability, complexity and flexibility. To measure these high-level quality attributes design properties like modularity, cohesion, coupling and encapsulation are used. This model is divided in four levels. First level consists of design quality attributes. Second level contains design properties of object-oriented design. Object oriented design metrics fall in level three and design components of object-oriented design are placed in level four. Authors identified following design quality attributes:

1. Re usability
2. Flexibility
3. Understandability
4. Functionality
5. Extendibility
6. Effectiveness

These quality attributes are not tangible means cannot be observed or measured directly. There is no direct way that can tell that a source code is re-usable or not, or code is flexible or not.

To analyze a quality attribute, it is mapped with a set of object-oriented design properties. Design properties can be measured or observed using functionality, relationship and structure of design components. This model identified following design properties:

1. Abstraction
2. Encapsulation

3. Coupling
4. Cohesion
5. Composition
6. Inheritance
7. Polymorphism
8. Messaging
9. Complexity
10. Hierarchies

These design properties are measured using object-oriented metrics. Table 5 Contains the metrics used to calculate quality attributes. Design components are identified as classes, objects, relationships, hierarchies, methods and attributes. [32]

Table 2.5 QMOOD Metrics

Metric	**Quality Attribute**	**Explanation**
DSC (Design Size in Classes)	Design Size	Total number of classes in design
NOH (Number of Hierarchies)	Hierarchies	Count of Hierarchies in design
ANA (Average number of Ancestors)	Abstraction	Same like DIT in CK Metrics
DAM (Data Access Metric)	Encapsulation	Ratio between private attribute and total attributes
DCC (Direct Class coupling)	Coupling	Number of directly connected classes
CAM (Cohesion Among Methods)	Cohesion	Count of parameters that are common between

		method attributes and independent parameters
MOA (Measure of Aggregation)	Composition	Count of user defined data types
MFA (Measure of Functional Abstraction)	Inheritance	Ration of inherited methods of class to total accessible methods of class
NOP (Number of Polymorphic methods)	Polymorphism	Number of virtual methods
CIS (Class Interface Size)	Messaging	Count of public methods
NOM (Number of Methods)	Complexity	Count of all methods in class

2.3.7 Coupling Metrics

Coupling is very important in code quality and various metrics have been proposed for its measurement. This sections briefly describes the important coupling metrics.

MPC

Message Passing Coupling metric depicts complexity of messages transferred among different classes. It can be calculated by calculating number of send statements. This metric indicates dependency of class methods on other classes.

RFC

Response for Class is metric that counts the members of response set for a class. Response set consists of all methods that are locally defined in a class plus all methods that are called by local methods of a class. Size of response set can indicate complexity of class.

DAC

Data Abstraction Coupling is a measure of coupling created through abstract data types. If a class A has a property of type B means A contains a property x of type class B, this is DAC coupling as class A can access all data and methods of class B via property x. DAC is calculated by counting number of ADTs used in a class [33]

2.3.8 Cohesion Metrics

TCC

Tight Class Cohesion measures cohesion based on direct connections among method pairs and maximum possible number of pairs of methods

LCC

Loose Class Cohesion measures cohesion based on direct plus indirect connections among method pairs and maximum possible number of pairs of methods [34]

2.4 Software Quality Frameworks and Tools

Measuring source code quality is a challenge and many researchers have worked on proposing different quality measurement frameworks. Also, there are many tools available that calculate different code metrics to provide quality information about source code. Some of the frameworks and tools are discussed in this section to give an overview of tools and frameworks proposed for code quality analysis.

2.4.1 Maintainability Index (MI)

Maintainability index is combination of few metrics. It is combination of LOC, Volume proposed by Halstead and cyclomatic complexity metric by McCabe. It is calculated as:

Equation 2 Maintainability Index

$$MI = 171 - 5.2ln(V) - 0.23V(g) - 16.2ln(LOC)$$

here V is volume and V(g) is complexity [35]

Maintainability index can be used as quality indicator. Higher values show that it is easy to maintain code. Three ranges are defined high, medium and low for MI. 0-64 is range for low MI. 65-84 indication of medium maintainability and 85-118 as high maintainability [11]

2.4.2 Source Code Quality Framework for C languages

This framework was proposed to analyze code quality of C language-based programs. The framework consists on metric calculation tool, metric value normalization tool, aggregation tool and a tool for generating rating levels along with few standard rating levels for each metric. This framework provides analysis of portability, maintainability, reliability and reuse ability of code in quantitative form. This framework used QAC and LogiScope tools for calculating code metrics. Although this framework provides quality analysis in terms of quality factors specified by ISO9126 but it is not making use of most popular metrics proposed in literature for object-oriented code. [36]

2.4.3 Intelligence Code Evaluator

Intelligence Code Evaluator is tool for Java source code for analyzing code quality based on metric values. Sequencer, syntax analyzer, metric analyzer and evaluator are basic components used to analyze Java code. This tool used only few metrics from literature to measure code quality. Lines in class, number of methods in class and number of classes which have default constructor are metrics used in this tool in metric analyzer module [37]

2.4.4 Designite

Designite is code evaluation tool that analyzes quality through code smells which appear at design level. This tool is implemented in C#. Code is parsed via another tool named NRefactory. NRefactory creates AST from Parsed code. Proposed tool use AST to create a meta-model of hierarchical type. This meta-model contains project objects. Project object has namespace objects that are part of project. And namespace object contains class/type objects. This tool takes C# solution file as input and generates output using some metrics. 30 popular metrics are used to detect design smells. This tools also

detects code clones. After evaluation of code results are shown based on metrics which re violated and smell found in code. Design smells are detected from meta-model.

Code metrics are used at method, type and project level. Tool provide facility to select a metric and view project quality according to selected metric perspective. All entities of a project are divided in four categories based on every metric value. This categorization is shown in form of pie chart. Each pie slice represents a category. These four categories are below or equal to threshold, slightly above threshold, quite above threshold, dangerously above threshold.

Designite evaluates dependency among namespaces, types and even projects. It further provides hotspot analysis of code smells. In hotspot analysis it highlights a subset of classes that have 20% code smells. This helps in detecting classes that are responsible for low quality and can be re factored to improve code. This tool also provides clone detection. If code block of more than 20 lines is repeated it is counted as clone. [38]

2.4.5 QualityGate SourceAudit

SourceAudit tool measures the maintainability of software code using standards defined in ColumbusQM model of ISO/IEC 25010. This tool measures the maintainability of source code using metrics and then aggregating metrics to high level elements. Maintainability is very important quality factor due to its direct link with development cost. If code has higher maintainability its development cost is low and vice versa. This tool analyzes code using benchmarks.[39]

2.4.6 E-Quality

This tool represents quality of software visually using graphs. It calculates quality metrics and relations between class from code. It represents results in in form of 2D

graphs which are helpful in analyzing issues in design of software[40]. This tool is developed as Eclipse tool for quality analysis of Java code. It picks class relations and metrics from code and represent them in interactive graphs. This tools shows a software as diagraph with labels, G(V, E) where V represents the vertices that denote to classes of software or interfaces of software; E shows the of edges which denote to the relations between lasses. Weights are assigned to edges based on strength of between two classes/interfaces. Coupling, cohesion, size and complexity metrics are calculated and shown on graph using different shapes, colors and filling of shapes. Size of shape increase with increase of size of class. Shape edges increases with increase in coupling of class. Color of shaped changed from green to orange as complexity of class increases. Filling of shape becomes dense as cohesion of class decreases. Visual representation of this tool becomes very complex with increase in size of software.

2.4.7 PHP_depend

Php_depend is PHP based static code analyzing tool. It analyzes coupling, complexity, inheritance and size of software code. It uses OO code quality metrics like LOC, NOM, NOP (Number of Packages), NOC (Number of Classes), CYCLO (Cyclomatic complexity number), CALLS (method calls that are distinct), FANOUT, AHH (Average hierarchy Height) and ANDC (Average number of derived classes).

This tool shows results in form of pyramid. Inheritance metrics are shown on top, right half contains coupling metrics and left half shows size and complexity metrics. This tool is quite useful but it does not depict quality at higher level.[41]

2.4.8 CodeMR

CodeMR is tool for static analysis of code for Java, Scala and C++. It use code metrics and quality factors for evaluation of code. It measures coupling, complexity and

size of software code. Features of this tool include model editor, graph editor, quality attributes, module extraction and report generation. Different symbols are used to denote coupling, complexity and size. This tool includes package metrics, class metrics and method metrics. [42] Visual display of results helps in getting the big picture of code quality.

2.4.9 PhpMetrics

This tool is developed in PHP to analyze source code written in PHP. It uses various procedural and object-oriented metrics to calculate complexity and instability of code. The results are represented visually to make analysis easy [43]. PhpMetrics calculates size complexity and abstractness of code. It also shows count of bugs count in every class. This tool also calculates afferent coupling, efferent coupling and instability on class level. It can be used for static analysis of PHP code for coupling and class relation analysis. It shows results in textual as well as in graphical format. Results are shown at project level, package level and at class level. Only few object-oriented metrics are used to calculate size, complexity coupling and abstractness of code. High level quality factors like change proneness, testability etc. are not measured in this tool

2.5 Summary

Analysis of software quality is as old as history of software. Researchers worked on different dimensions for suggesting an applicable solution for software quality analysis. Quality of software is analyzed based on attributes, characteristics or factors that represent quality. Correctness, portability, reliability, maintainability, reusability, efficiency, flexibility, interoperability, integrity, usability and testability are important attributes of software quality.

Review of popular quality models like ISO 9126, McCall, Boehm, FURPS and Dromey's revealed that software quality can be measured using quality factors.

Factors are broken down in criteria and criteria is measured using metrics. Quality factors are grouped to analyze quality from different perspectives. Many metrics have been proposed for measuring different aspects of software code. MOOD, QMOOD and MOOSE metric suits are most popular for object-oriented code quality. Many frameworks and tools have been proposed for software code quality analysis. Some frameworks proposed methods to calculate quality in quantitative way but quality factors are not measured. Some frameworks only measure Maintainability or reliability of software

CHAPTER 3

PROPOSED METHODOLOGY

This section will describe the process used to define the framework for code quality analysis and present calculated quality in quantitative format.

3.1 Code Paradigm

It has been revealed through study of code metrics that object-oriented code is most studied paradigm in terms of quality. Most popular metrics are proposed for object-oriented code and this paradigm is most popular in software industry. Based on this fact it has been decided that suggested framework will concentrate on object-oriented code. In object-oriented programming code is divided in entities called classes. Classes contains data and methods that operate on that data. OOP provides encapsulation, inheritance and polymorphism features which are not present in structural programming.

3.2 Code Quality Factors and Metrics

Software quality models like McCall model, Boehm's model and ISO 9126 distributed quality in characteristics or factors. McCall model broken down quality in maintainability, reusability, portability, reliability, interoperability, correctness, efficiency, integrity, testability, flexibility and usability.

Source code quality is a bit different from general software quality. As code quality checks only internal quality of software. Some quality factors like usability cannot be measured from code so we are not including this factor in code quality evaluation. Suitable quality factors for measuring quality of source code are given below:

1. Reliability
2. Maintainability
3. Testability
4. Reusability
5. Portability
6. Understandability
7. Simplicity
8. Auditability[7]

In 2016 a group of researchers conducted a study to analyze relationship between factors of code quality and quality metrics for code. It has been found that many of the quality factors can be measured with some metrics that have been proposed for object-oriented code. We have used mappings of this study as basis for new framework.

Table 3.6 Quality Factors Mapping with Metrics

Code Quality Factor	**Source code metric**
Maintainability	DIT
	LOC
	WMC
	CC-VG
	TCC
	NOCC
	RFC

	MPC
	DAC
	NOM
Reusability	LCOM
	LOC
	CBO
	RFC
	MPC
	WMC
	NOCC
Change Proneness	DIT
	NOCC
	CBO
	RFC
	LCOM
	DAC
	NOA
Stability	WMC
	LOC
Testability	RFC
	CBO

	LCOM
	LOC

[44]

All metrics used in mappings are studied in detail from literature. Detailed analysis revealed that some of mapped metrics definition cannot be found in literature so those metrics are dropped from list of selected metrics. These dropped metrics are External Class Complexity, External Class Size and System Design Stability.

We have selected following five quality factors for new framework:

- Maintainability
- Reusability
- Change proneness,
- Testability
- Stability

understandability and Modifiability also have metrics mappings but we these two factors as they fall in sub category of maintainability and metrics in these two quality factors also appeared in with maintainability mapping. Testability is considered as separate factor despite being a sub factor of maintainability in suggested framework because its mapped metrics are different from maintainability metrics. Also, testability is important code quality.

3.3 OOCQM

This study suggests a framework Object Oriented Code Quality Meter (OOCQM) for calculating source code quality in quantitative form. Quality is calculated in terms of

code quality factors. This framework is specially designed for object-oriented code. This framework is shown in figure 3.1. Main components of OOCQM are:

- Read and parse code
- Metrics calculation and normalization
- Factors calculation
- Quality calculation

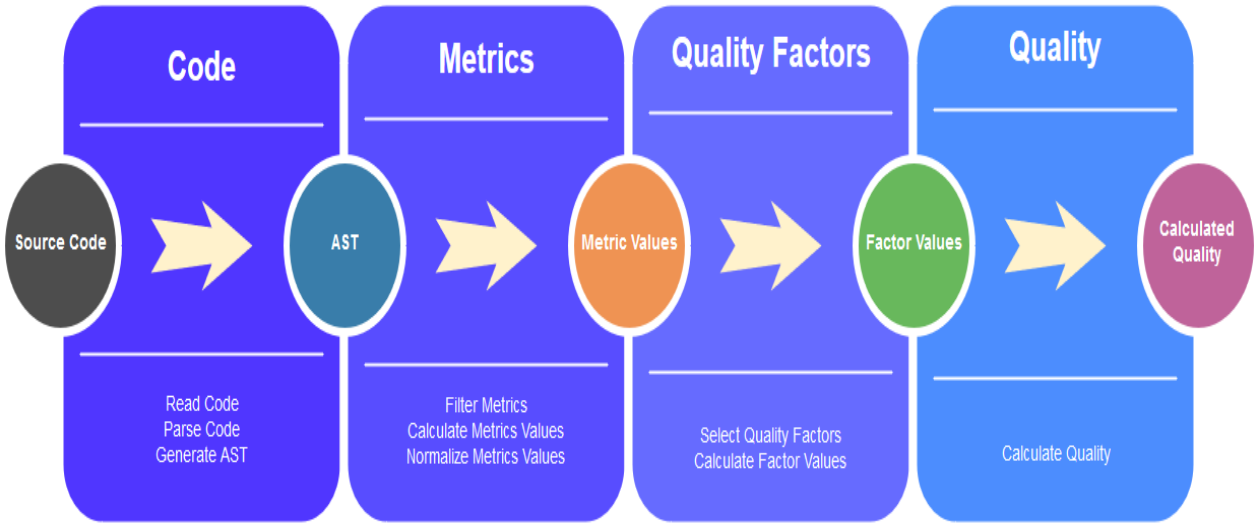

Figure 3.1 OOCQM Framework

Each of these components or modules is discussed below in detail

3.3.1 Read and Parse Code

This component reads object-oriented code of selected project. All classes present in project are read one by one. Size of project can vary from few classes to hundreds of classes/files. A language specific parser (Nikic for PHP) is used to parse code to generate Abstract Syntax Tree (AST). AST allows calculation of metrics in a convenient way. An AST is a representation source code structure in form of tree of any programming language. Each construct is represented by a node in tree. All statements are broken down in trees. After reading code each class is represented as tree of nodes.

3.3.2 Metrics Calculation and Normalization

Definition of every selected metric is traced from literature and then calculated according to definition. A database is used to save intermediate values that are used in some metric calculations. Database also saves metric values based on class, based on quality factor and based on project

Different metric values have different ranges like DIT has value n where n can be any positive integer including zero. While value of metric TCC lies between 0 and 1. Metric values are evaluated in most studies through thresholds. But thresholds approach has following shortcomings:

- Thresholds are not available for all metrics
- Threshold values for one metric vary in different studies so difficult to choose one threshold.
- Threshold values are dependent on other factors like code size, code type etc.

This study suggests to normalize each metric value using min max normalization technique. First of all, metric values of all classes are calculated then these values are normalized using minimum and maximum value of that metric. Let's say there are N classes in a project and M is set of all values of a metric m. Normalized Value V` of a class C calculated as:

Equation 3 Min Max Normalization

$$V`(Cm) = V - max(M) / max(M) - min(M)$$

Here V is the actual value of metric m for class C. This method makes all values of all metrics fall in range of 0-1.

After calculating and normalizing metrics values for all classes. Project base metric value is calculated by calculating average metric value of all classes. For a project P metric value calculation can be described as given below:

Equation 4 Metric Calculation for Project

$$V\ (Pm)\ =\ \frac{\sum_{i=1}^{n}(C_m)}{n}$$

Here n is count of classes in project. Cm is metric value for class C.

3.3.3 Quality Factors Calculation

Five quality factors are chosen for OOCQM. Only those quality factors are selected for which code metrics exist in literature. Each factor assigned 20 points so the sum of all factors is 100. All selected factors maintainability, change proneness, reusability, testability and stability are given equal points as all of these are equally important for a good quality software. The value of each quality factor is calculated based on all metric values that are mapped against that factor. It is tested that all metrics that are mapped against a quality factor has negative correlation with selected factor. For example, let's have look at metric WMC. It is mapped against maintainability, reusability and stability. Increase in value of WMC decreases these three factors. To create a relationship between metrics and quality factors that can depict this negative correlation following equation is devised.

Equation 5 Factor Calculation for Project

$$F\ =\ \sum_{i=1}^{n}(20/n) * (1 - V\ )$$

Here F is quality factor value, n is number of mapped metrics against selected factor. V

is average value of metric.

3.3.4 Quality Calculation

Calculated values of all five quality factors are aggregated up to make a numerical value between 1- 100. It can be described as given below.

Equation 6 Quality calculation for Project

$$Q = \sum_{i=1}^{n} (f(v))$$

Q is quality of project. N is count of quality factors in framework. f (v) is value of selected factor

3.4 OOCQM Tool

In this section the main focus of discussion is on tools and technologies used to implement OOCQM framework as a tool so it can be tested with real programming code. Selection of tools and technologies for development of tool was dependent on coding capabilities of researcher. In research-based tools development researcher should choose technology in which one is proficient or that technology is easy to learn so research time is not wasted in tool development and focus of study is not disturbed

3.4.1 Tools and Programming Languages Used

Following tools and languages are used in the implementation of OOCQM implementation:

- Laravel
- MySQL
- Nikic PHP Parser

3.4.1.1 Laravel

Laravel is one of the most popular web development frameworks for PHP language. It

is free and open source. Laravel was created by Taylor Otwell. Architectural structure of Laravel is based on MVC(Model-View-Controller) pattern. Laravel has packaging system that is modules based and it has a dedicated dependency manager. Laravel makes development lifecycle simple by easing common things used in web projects, like:

- Fast and simple routing engine
- Container for dependency injections
- Multiple options for cache and session storage
- Expressive database ORM
- Database seeding and migrations
- Event broadcasting at real time

3.4.1.2 MySQL

MySQL is a relational database system that is free and open source. It is an important element of LAMP stack. It is lightweight and robust database management system. Top features of MySQL are:

- Good performance with scalability so the need of increasing data loads can be fulfilled.
- MySQL provides clusters that provide self-healing replication to improve performance, scalability, and availability.
- Changing schema online to fulfill changing requirements of business.
- Schema for performance to monitor performance at user- level and application-level and monitor resource consumption.
- Structured Query Language and No SQL access for executing complex queries and easy and fast operations based on key value.
- It provides Platform Independence which gives the flexibility for developing and deploying on different operating systems.
- It allows for Interoperability of Big Data using this RDBMS as it can provide the operational data store for Cassandra and Hadoop.

3.4.1.3 Nikic Parser

Programming language parser provides static analysis of the code. Proposed framework is developed as tool to verify the results of proposed framework. Initially this tool is developed for evaluation of object-oriented code written in PHP.

Nikic parser is used in this tool for code parsing. The main features of Nikic parser are

- It can parse all versions of PHP ranging from 5.2 to 7.4
- It parses code to AST
- It can identify invalid code that is not written according to language rules
- Generated AST shows code in from of nodes
- AST can be displayed in a format that is readable for humans
- Nikic can also convert AST to PHP code
- It can also create AST from JSON and vice versa
- It can evaluate constant expressions in code

3.5 Summary

Quality factors are picked based on mapping available in literature with metrics. Code is parsed and selected metrics are calculated then values of metrics are normalized. Normalized metric values are aggregated to derive a factor value. Factor values are further aggregated to generate a numerical value for quality. Nikic is used as parser in many modern PHP frameworks like Laravel. Source code metrics are derived from ASTs that are generated after code parsing. Some of the metrics like LOC or NOM are calculated directly from ASTs. While for calculation of other metrics calculation database is used for calculations of intermediate variable. OOCQM is developed in Laravel.

Formulas for calculation of selected metrics are picked from literature where these metrics were suggested. Class level metrics are used in this framework. To handle the variation in code length of a class metric values are normalized using min-max normalization between 0 to 1 inclusive.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 OOCQM Validation

Validation of proposed framework is very important because without validation it is not possible to prove its usefulness for research community, software industry and academics.

4.1.1 Read and Parse Code

Here are two sample classes written in PHP for which we will apply parsing in AST

```php
<?php
namespace mycode;
class Shape
{
    protected $name;
    protected $type;

    public function draw(){
        echo 'Draw Shape';

    }
    public function __construct($name, $type)
    {
        $this->name = $name;
        $this->type = $type;


    }

}
class Circle extends Shape
{
    private $radius=10;
    public function getRadius(){
        return $this->radius;
    }
    public function draw()
    {
        parent::draw();
        echo 'Circle';
    }

}
```

AST of Code

```
array(
    0: Stmt_Namespace(
        name: Name(
            parts: array(
                0: mycode
            ) )
        stmts: array(
            0: Stmt_Class(
                flags: 0
                name: Identifier(
                    name: Circle
                )
                extends: Name(
                    parts: array(
                        0: Shape
                    ) )
                implements: array( )
                stmts: array(
                    0: Stmt_Property(
                        flags: MODIFIER_PRIVATE (4)
                        props: array(
                            0: Stmt_PropertyProperty(
                                name: VarLikeIdentifier(
                                    name: radius
                                )
                                default: Scalar_LNumber(
                                    value: 10
                                ) ) ) )
```

```
1: Stmt_ClassMethod(
        flags: MODIFIER_PUBLIC (1)
        byRef: false
        name: Identifier(
            name: getRadius
        )
        params: array(
        )
        returnType: null
        stmts: array(
            0: Stmt_Return(
                expr: Expr_PropertyFetch(
                    var: Expr_Variable(
                        name: this
                    )
                    name: Identifier(
                        name: radius
                    )
                )
            )
        )
    )
```

```
2: Stmt_ClassMethod(
        flags: MODIFIER_PUBLIC (1)
        byRef: false
        name: Identifier(
            name: draw
        )
        params: array(
        )
        returnType: null
        stmts: array(
            0: Stmt_Expression(
                expr: Expr_StaticCall(
                    class: Name(
                        parts: array(
                            0: parent
                        )
                    )
                    name: Identifier(
                        name: draw
                    )
                    args: array(
                    )
                )
            )
            1: Stmt_Echo(
                exprs: array(
                    0: Scalar_String(
                        value: Circle
                    )
                )))
    )))
```

4.1.2 Metrics Calculation

Metric Values for "Circle" Class

Table 4.1 Metric Values for Circle Class

Metric Name	Metric Actual Value	Normalized Value
DIT	1	1
LOC	14	0.9
WMC	1	1
CC-VG	1	1
LCOM	2	1
TCC	0	0
NOCC	0	0
RFC	8	1
MPC	0	0
DAC	2	1
NOM	2	1
CBO	2	1
NOA	1	0.5

Table 4.1 shows calculated values and normalized values for all selected metrics for Circle class

4.1.2 Quality Factors Calculation

Quality factor values are calculated at class level than aggregated to get over all factor value for project. Following table shows factor calculations for class "Circle":

Table 4.2 Factor Values for Class "Circle"

Factor Name	Factor Value
Maintainability	0.12
Reusability	0.19
Change Proneness	1.4
Testability	0.3
Stability	0.6

4.1.2 Quality Calculation

Quality factors are aggregated to calculate overall quality value.

Table 4.3 Quality Calculation

Factor Name	**Factor Value**
Maintainability	7.10
Reusability	6.16
Change Proneness	6.78
Testability	3.29
Stability	0.3
Quality	**23.68**

4.2 OOCQM's Results Comparison with MI Results

In order to test the OOCQM tool different versions of Laravel are chosen. Laravel is considered one of the best PHP frameworks for developing web applications. Five versions of Laravel are chosen to apply OOCQM. This tool is executed on selected Laravel versions one by one and results of metrics, quality factors and overall quality are recorded. Results of OOCQM are shown in Table 4.1.

In order to prove the correctness of OOCQM framework its results need to be compared with a measure that has been proved authentic in research community. MI is chosen as comparative measure for the results of this framework. Results of OOCQM are compared with MI as shown in Table 4.1. First column shows project name, second column shows maintainability, third column shows reliability, fourth column shows change proneness, fifth column testability, sixth column stability and seventh column shows overall quality of project. MI values of selected versions of Laravel are calculated and shown in last column on Table 4.1. We have evaluated that quality values generated by OOCQM are in accordance with MI values for different versions of Laravel. Laravel version 5.1 has highest value for maintainability index and this version has highest value of quality as well. Laravel 5.3 has lowest value for both quality and

maintainability index. If MI value is higher for a any version of Laravel OOCQM quality value of also higher. If MI value is lower OOCQM quality value is also lower

Table 4.4 OOCQM Results

Project	***Maintainability***	***Reusability***	***Change proneness***	***Testability***	***Stability***	***Quality***	***MI***
Laravel 5.1	12.75	14.03	12.63	12.99	10.27	62.68	194.95
Laravel 5.2	11.47	13.69	11.83	11.7	11.97	60.69	193.53
Laravel 5.3	12.16	13.18	11.92	11.92	10.2	59.63	191.3
Laravel 5.4	12.41	13.48	12.15	12.32	10.06	60.45	192.6
Laravel 5.5	12.45	13.76	11.82	12.54	10.44	61.02	193.5

MI values of higher than 118 are considered very good. This suggests that all selected version of Laravel have good quality. The above table shows that Laravel 5.2 is more stable version than other versions listed in the table. Results listed in above table also indicate that value of every quality factor will impact the overall quality. If any factor has a low value it will lower the overall quality of that project. This was the comparison of OOCQM with MI. In second part of OOCQM validation the features of OOCQM are compared other frameworks or tools.

Table 4.5 OOCQM comparison with other tools

Tool Name	***Quality Factors Measured***					***Overall Quality***
	Maintainability	***Testability***	***Reusability***	***Change Proneness***	***Stability***	
SCQFC [16]	Y	N	Y	N	N	N
Designite [18]	N	N	N	N	N	Y
PHP_depend [20]	N	N	N	N	N	N
SourceAudit [19]	Y	N	N	N	N	N
ICE [17]	N	N	N	N	N	Y
OOCQM	Y	Y	Y	Y	Y	Y
CodeMR [21]	N	N	N	N	N	Y
PHPMetrics [22]	Y	N	N	N	N	Y

4.3 OOCQM Comparison with Other Tools

We compared OOCQM with some other quality analysis tools s shown in Table 4.2. Most of these tools are not calculating code quality in terms of quality factors. SCQFC measures quality in terms of reliability, maintainability and reusability but it does not calculate quality as a numerical value. PhpMetrics calculates only maintainability of source code.

4.4 Summary

Calculation of code quality of different versions of PHP framework Laravel using OOCQM and then comparing those results with maintainability index value revealed that framework results are in accordance with MI values. In second phase of

comparison OOCQM is compared with few other tools that quantify code quality of source code. This comparison shown that OOCQM is the only tool that calculates code quality using five quality factors

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Introduction

After validating the results of OOCQM the last part of this study is to summarize this study. This chapter will provide a highlight the areas where this framework is applicable. After discussion of applications of OOCQM a brief discussion is provided for future work on this framework. Last section of this chapter provides the summary of whole research study.

5.2 Applications of OOCQM

OOCQM tool can be used in software industry for measuring different quality factors of a source code and overall quality as a whole. This framework measures quality in a reasonable way for projects regardless of its size. So measuring quality of different applications of variant sizes is possible. This tool can be helpful for both developers and managers. Developer can check how reliable and stable their code is. Managers can predict the reusability and maintainability of any project using this tool. This tool also can be used in academics for analyzing coding skills of students by testing their code assignments.

5.3 Future Work

Currently this framework is generic but tool is only developed to analyze object-oriented code of PHP language. This can be enhanced to analyze code of other programming languages by integrating parsers of those languages.

Quality value generated by this tool ranges from 0 to 100 where lower value means lower quality and higher value means higher quality. Further work is required to decide threshold or ranges for low, medium and high quality.

5.4 Conclusion

OOCQM is a generic code quality analyzer for object-oriented code. It consists on popular metrics some of which are never used collectively before in any tool for quality analysis of source code. This framework measures different aspects of object-oriented code like Size, complexity, cohesion, coupling and abstractness of code and converts these measurements into numerical value of quality. It calculates quality factors which are not calculated before using metrics. We also used some metrics like RFC, DAC in calculation of quality factors which are never used in any framework previously. Comparison of this framework's results with MI has proved the validity of it. Analysis of OOCQM features with other tools has shown that OOCQM is the only framework that provides numerical values for both overall quality and quality at factor level. In future it can be extended for more programming languages. Quality value threshold is an aspect that can be further explored that which quality number is good and which is not good.

References

[1] J. P. Cavano and J. A. McCall, “A framework for the measurement of software quality,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 3, no. 5, pp. 133–139, 1978.

[2] B. Kitchenham and S. L. Pfleeger, “Software quality: the elusive target,” *IEEE Softw.*, vol. 13, no. 1, pp. 12–21, 1996.

[3] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, “Software quality models: Purposes, usage scenarios and requirements,” *Proc. - Int. Conf. Softw. Eng.*, pp. 9–14, 2009.

[4] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, “Source code metrics: A systematic mapping study,” *J. Syst. Softw.*, vol. 128, pp. 164–197, 2017.

[5] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” *Proc. 2nd Int. Conf. Softw. Eng.*, pp. 592–605, 1976.

[6] G. F. McCall, Jim A and Richards, Paul K and Walters, “Factors in software quality, volumes I, II, and III,” *US Rome Air Dev. Cent. Reports, US Dep. Commer. USA*, vol. I, no. November, 1977.

[7] T. Iqbal, M. Iqbal, M. Asad, and A. Khan, “A Source Code Quality Analysis Approach.”

[8] J. Mooney, “Bringing portability to the software process,” *Dept. Stat. Comp. Sci., West Virginia Univ., Morgant. WV*, pp. 559–570, 1997.

[9] B. Kitchenham, “Towards a Constructive Quality Model: Part I: Software Quality Modelling, Measurement and Prediction.,” *Softw. Eng. J.*, vol. 2, no. 4, pp. 105–113, 1987.

[10] A. K. Bain and P. Chand, “Ferroelectrics: Principles and Applications,” p. 328, 2017.

[11] R. Anggrainingsih, B. O. P. Johannanda, A. P. Kuswara, D. Wahyuningsih, and T. Rejekiningsih, “Comparison of maintainability and flexibility on open source LMS,” *Proc. - 2016 Int. Semin. Appl. Technol. Inf. Commun. ISEMANTIC 2016*, pp. 273–277, 2017.

[12] P. Wegner and P. Wegner, “Interoperability,” *ACM Comput. Surv.*, vol. 28, pp. 285--287, 1996.

[13] J. C. Strauss and K. J. Thurber, “Considerations for new tactical computer systems,” *Proc. - Int. Symp. Comput. Archit.*, pp. 135–140, 1977.

[14] X. Ferré, N. Juristo, H. Windl, and L. Constantine, “Usability basics for software developers,” *IEEE Softw.*, vol. 18, no. 1, pp. 22–29, 2001.

[15] S. Editor *et al.*, “Software Metrics Chapman & Hall/CRC Innovations in Software Engineering and Software Development Software Test Attacks to Break Mobile and Embedded Devices Software Designers in Action: A Human-Centric Look at Design Work Introduction to Combinatorial Testing.”

[16] “Project Quality Management - Belmarie Prado bsit3c - Medium.” [Online]. Available: https://medium.com/@bingbingpenacilla/project-quality-management-506f43aab409. [Accessed: 06-Oct-2019].

[17] I. Heitlager, T. Kuipers, and J. Visser, “A Practical Model for Measuring Maintainability,” *6th Int. Conf. Qual. Inf. Commun. Technol. (QUATIC 2007)*, pp. 143–153, 2007.

[18] A. B. Al-Badareen, M. H. Selamat, M. A. Jabar, J. Din, and S. Turaev, “Software quality models: A comparative study,” *Commun. Comput. Inf. Sci.*, vol. 179 CCIS, no. PART 1, pp. 46–55, 2011.

[19] “Software Quality Management.” [Online]. Available: http://sce2.umkc.edu/BIT/burrise/pl/software_quality_management/. [Accessed: 06-Oct-2019].

[20] P. Rafa E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and Comparative Study,” *J. Am. Sci.*, vol. 6, no. 3, pp. 166–175, 2010.

[21] K. Musa and J. Alkhateeb, “Quality Model Based on Cots Quality Attributes,” *Int. J. Softw. Eng. Appl.*, vol. 4, no. 1, pp. 1–8, Jan. 2013.

[22] D. Samadhiya, S. H. Wang, and D. Chen, “Quality models: Role and value in software engineering,” *ICSTE 2010 - 2010 2nd Int. Conf. Softw. Technol. Eng. Proc.*, vol. 1, pp. 320–324, 2010.

[23] D. Jamwal, “Analysis of Software Quality Models for Organizations,” *Int. J. Latest Trends Comput. (E*, vol. 1, no. 2, pp. 2045–5364, 2010.

[24] “(PDF) Quality Models in Software Engineering Literature: An Analytical and Comparative Study.” [Online]. Available: https://www.researchgate.net/publication/228991952_Quality_Models_in_Software_Engineering_Literature_An_Analytical_and_Comparative_Study/figures?lo=1. [Accessed: 06-Oct-2019].

[25] H. Zulzalil, A. Abd Ghani, M. Selamat, and R. Mahmod, “A case study to identify quality attributes relationships for web-based applications,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 11, pp. 215–210, 2008.

[26] D. Nabil, A. Mosad, and H. A. Hefny, “Web-Based Applications quality factors: A survey and a proposed conceptual model,” *Egypt. Informatics J.*, 2011.

[27] J. Rosenberg, “Some Misconceptions About Lines of Code,” *Proc. Fourth Int. Softw. Metrics Symp.*, pp. 137–142, 1997.

[28] A. Of and T. Designs, “Halstead ’ s metrics: analysis of their designs 1,” pp. 145–159, 1977.

[29] T. J. McCabe, “A Complexity Measure,” *IEEE Trans. Softw. Eng.*, vol. SE-2,

no. 4, pp. 308–320, 1976.

[30] S. R. Chidamber, C. F. Kemerer, S. R. Chidamber, and C. F. Kemerer, "Towards a metrics suite for object oriented design," *ACM SIGPLAN Not.*, vol. 26, no. 11, pp. 197–211, Nov. 1991.

[31] R. Harrison, S. J. Counsell, and R. V. Nithi, "An evaluation of the MOOD set of object-oriented software metrics," *IEEE Trans. Softw. Eng.*, vol. 24, no. 6, pp. 491–496, 1998.

[32] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, 2002.

[33] S. Li, W., & Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Syst. Softw.*, vol. 23(2), pp. 111–122, 1993.

[34] L. M. Ott and J. M. Bieman, "Program slices as an abstraction for cohesion measurement," *Inf. Softw. Technol.*, 2002.

[35] K. D. Welker, "The Software Maintainability Index Revisited," no. August, 2001.

[36] H. Washizaki, R. Namiki, T. Fukuoka, Y. Harada, and H. Watanabe, "A Framework for Measuring and Evaluating Program Source Code Quality," in *International Conference on Product Focused Software Process Improvement*, 2007, pp. 284–299.

[37] M. Sangeetha, C. Arumugam, K. M. Senthil Kumar, and P. S. Alagirisamy, "Enhancing internal quality of the software using intelligence code evaluator," *Commun. Comput. Inf. Sci.*, vol. 330 CCIS, pp. 502–510, 2012.

[38] T. Sharma, "Designite - A Software Design Quality Assessment Tool," pp. 1–4, 2016.

[39] T. Bakota, G. Ladányi, and R. Ferenc, "QualityGate SourceAudit : A Tool for Assessing the Technical Quality of Software," pp. 440–445, 2014.

[40] Erdemir, Ural, Umut Tekin, and Feza Buzluca. "E-Quality: A graph based object oriented software quality visualization tool." 2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT). IEEE, 2011.

[41] R. Randriatoamanana, “Object Oriented Metrics to measure the quality of software upon PHP source code with PHP _ depend Study case Request online System application,” pp. 2–6, 2017.

[42] “CodeMR Static Code Analysis Tool.” [Online]. Available: https://www.slideshare.net/codemr/codemr-software-quality. [Accessed: 04-May-2019].

[43] “Metrics of PhpMetrics.” [Online]. Available: http://www.phpmetrics.org/documentation/index.html. [Accessed: 26-May-2019].

[44] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, P. Avgeriou, and E. M. Arvanitou, “A Mapping Study on Design-Time Quality Attributes and Metrics Elvira,” *J. Syst. Softw.*, 2017.