2008:PR001

UNIVERSITY WEST

Software Metrics – Usability and Evaluation of Software Quality

Muzamil Jah

MASTER'S THESIS
Software Engineering, 2008
Department of Technology, Mathematics and Computer Science

MASTER'S THESIS

Software Metrics – Usability and Evaluation of Software Quality

Abstract

It is difficult to understand, let alone improve, the quality of software without the knowledge of its software development process and software products. There must be some measurement process to predict the software development, and to evaluate the software products. This thesis provides a brief view on Software Quality, Software Metrics, and Software Metrics methods that will predict and measure the specified quality factors of software. It further discusses about the Quality as given by the standards such as ISO, principal elements required for the Software Quality and Software Metrics as the measurement technique to predict the Software Quality. This thesis was performed by evaluating a source code developed in Java, using Software Metrics, such as Size Metrics, Complexity Metrics, and Defect Metrics. Results show that, the quality of software can be analyzed, studied and improved by the usage of software metrics.

Author:	Muzamil Jah		
Examiner:	Dr. Samantha Jenkins		
Advisor:	Dr. Samantha Jenkins		
Programme:	Software Engineering, 2008		
Subject:	Software Engineering	**Level**:	Master
Date:	June, 2008	**Report Number**:	2008:PR001
Keywords	Software Metrics, Software Product Metrics, Product Metrics, Software Quality Metrics, Evaluation of Software Code, Usability of Software Metrics.		
Publisher:	University West, Department of Technology, Mathematics and Computer Science, S-461 86 Trollhättan, SWEDEN Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se		

Acknowledgement

Dedicated to my parents
Dr. Rizwan Ahmed
and
Mahboob sultana Begum

I would like to express my profound gratitude and sincere thanks to my supervisor Dr. Samantha Jenkins (Assoc. Prof) for her constant help, guidance, encouragement, kind cooperation and trust throughout the thesis work. Under her supervision, I developed my abilities to identify and tackle research problems, and to build and present the research solutions in a comprehensible way. She always encouraged me to target top-level conferences and journals, which opened opportunities to obtain quality feedback and pointers to novel research directions for my studies.

Special acknowledgments are due for my friends who made my stay in Trollhattan-Sweden a memorable one. In particular, I am grateful to my friends from Karlstad University for their selfless support, good wishes and help.

Finally, my family deserves special recognition for their unconditional support during the past years of my life. I am eternally grateful to my parents for their love, innumerable sacrifices, patience and encouragement, to my brother Dr. Mukaram jah for rendering me the sense and the value of brotherhood, to my sister Nahid for her love, my brother in law Nazeeruddin and my cousin Khuddus Pasha for their support during my Ms study, and my niece Emaan for being such a joy in my life. I would like to thank my friends (Ravi and Satish) for always being there for me without their selfless services and support, I wouldn't have reached this far.

Contents

List of Tables

List of Figures

1. Introduction

Quality is always an issue on which most of the researchers are working on, while developing the software. With the increase in the software market, customers are expecting software's of higher quality and they are even willing to pay higher prices for the software. With this increase in expectations and hike in the software market, companies and countries are continuing to invest great deal of money, time, and effort in improving the software quality [1].

Software quality cannot be improved without knowledge of development process. The number of bugs and the errors occurred during the software development process have to be found in the early stages of development for better quality. If the errors are found late, then the corrective action will be very expensive [2] [3]. Software organisations will be greatly benefited if there is process to plan and predict the software development. The process of measuring the software is known as software metrics.

Software metrics is defined as, "an objective, mathematical measure of software that is sensitive to differences in software characteristics. It provides a quantitative measure of an attribute which the body of software exhibits [4]. Its aim is to know the development process of software by controlling the different aspects. So, it can be said that metrics are used to improve the ability to identify, control and measure the essential parameters of software during its development or it can also be said as measurement of software product and the process by which it is being developed.

The information gained from software metric can be used to manage and control the development process, which will lead to improvement in the results of the software product. Good software metrics must have the ability to predict the software development process. So some of the ideal properties of a software metrics are

- It must be simple, that is easily definable and clear.
- It must have an objective.
- It must be valid.
- It must be reasonable.
- It must be robust in nature.

If the usage of software metrics is proper, then it can even capture the complexity embedded in the program structure [5]. Depending upon the working software metrics are classified in two types. They are,

1. Product type metrics
   It measures the products of software. Ex: Source code and design documents.

2. Process type metrics.
   It measures the development of software process. Ex: Type of Methodology, overall development time, number of changes made to documents and number of bugs recovered during testing.

The results obtained from the software metrics can be used to indicate, which parts of software have to changed or modified. Software metrics have proved to reflect the software quality, and thus they have been widely used in software quality evaluation techniques [6] [7]. Software metrics are studied as a way to access the quality of large system [8] [9] and have been applied to object oriented systems as well [8] [10] [11].

This thesis discusses the quality as given by the standards such as ISO, importance of quality during the development of software, and lastly by choosing the Software Metrics, such as Size Metrics, Complexity Metrics and Defect Metrics to evaluate the Java code.

Roadmap

This thesis begins with the introduction explaining the need for the improvement in the software quality and software metrics as a process of measurement for software quality. Different types of metrics and their importance are further explained.

Section 2 presents the background, which is required to understand this thesis. The different views on quality, purpose of the software metrics, some examples of software metrics, and methodology for software metrics (as given by the IEEE) are explained in this section.

Section 3 describes the meaning of quality for different people, effects on quality with the increase in the complexity, and implementation techniques for software quality. The subsection in it explains the framework for software quality as provided by ISO. Then, definition of metrics and its implementation is explained in section 4. Software product metrics and some examples of product metrics, which are used to for the calculation of results is given in section 5.

Section 6 describes the method used in this thesis for its implementation and Section 7 presents the results of this thesis work that is calculated using the different types of metrics. The results are explained in discussion section, and conclusion derived from the results and its explanation is given in the conclusion.

Goals

The main goal of this thesis was to study, analyse and evaluate software metrics and their usage in improving the quality of the software. This was performed by evaluating a source code in Java using the various software metrics which are predefined. Apart from this, the thesis also includes,

- To study and discuss the various standards of quality such as ISO.
- To study and discuss the various software metrics using standards such as IEEE.

2. Background

Stakeholders in software success are those who will suffer the greatest losses from the software failure [12]. During its life cycle of software, different stakeholders view the quality of software in a different manner [13]. The developing team focuses on developing the software code, testing team focuses on the reducing the number of defects and customer care team focuses on the complete user experience and it views the quality as defect free operation which is easy to use, easy to install and user friendly. So, depending upon the person and his profession the definition of quality differs. Though there has been research going on for decades, researchers are still not clear with the measure of software quality, meaning of terms to describe the aspects of software quality and the degree to which the definition of software has to be included [13].

In an article titled ´The Failure of Quality by Kitchenham, she challenged the software industry in their approach for quality. She stated in her article, that quality systems and procedures have become a means to avoid blame rather that providing excellent product quality service and she concluded that "we need procedures that support efforts for producing quality software products rather than mindless bureaucracy that influences trust and goodwill between stakeholders"[14].

There have been attempts to develop the model for the measurement of software. Software metrics is an attempt to quantify some of the aspects of products generated during the development of software [15]. Its purpose is to make assessments of software life cycle, so as to confirm whether the software quality requirements are being met during the development. Software metrics have been proved to reflect software quality, and thus metrics have been widely used in software quality evaluation methods [16]. It has been believed that metrics should be included during the development of software.

There has been many software metrics which measure the design structure and complexity of the software system. The examples of the metrics which are primarily proposed to measure the source code are

- Halstead's software metrics
- McCabe's cyclomatic complexity metrics
- Kafura's information flow metrics
- Robillard's statement interconnection metrics
- Bail's HAC complexity and
- Adamov's hybrid metrics

IEEE has published a standard for the software quality metrics methodology [17], which led to the development in this field. Its aim was to provide a systematic approach for the establishment of software quality metrics by identifying, implementing, analyzing and validating the software quality metrics of a system. The development of metrics as given by IEEE is given in Table 1.

Table 1: IEEE Software Metrics Methodology [17]

Software Quality Activity	Development Phasing
Establishment of software quality requirements	-----
Identifying the software quality metrics	-------
Implementation of software quality metrics	----------
Analyzing the results of metrics	--------
Validating the metrics	------

3. Software Quality

Quality, it is difficult to define – not because of the difficulty to achieve, but because of the difficulty to describe the term. Quality has different meanings for different people, for example: if we ask some persons owning a car, then will define the quality as ruggedness of the car, or the fastness of the car, or the looks of the car. So, from this it can be said as the definition of the quality varies with the views of the person using or it can also be said as the view of the beholder.

When it comes to software, the beholder is, the person using the software, or the person interacting with the software, whenever it is executed. That is, the person will be satisfied when the software does what he or she wants it to do, when it is purchased. The software purchased includes the code, but the users will be only interested in the working and the services offered such as the user manual, help and support. In case of software developed for the internal use in a company or in an organization, the quality is about performance of the software whenever the user asks the development organization to produce it.

Quality cannot be defined by the technical excellence alone [18]. Quality of software gets affected by many human factors such as communication and motivation between the developers and testers, and the value of money for the development process. The developed software products and services must be affordable and the customer must be able to enjoy the usage of software.

Different people have different views on delivering software products with quality. Even though the developers produce software products with new features, but with flaws and with higher price, then it will be of no use. Quality of the software products must be defined according to the user's view, i.e., does the software perform as I wanted? If not, then the user concludes that the software is not of good quality. It is the concept that defines the quality of software – the degree to which the software product will fulfill the requirements specified. The requirements can be functional, non-functional, and it can also be requirements for maintenance, portability and so on. The importance of this concept is that the requirements of the product are the requirements for the quality [19]. And these requirements must be in such a way that the user wants it to be.

The ability to know what the user wants or expects from the software is the problem that affects the quality of software. Various studies show that 25% to 40% of defects in the software are caused due to the errors related to the requirements [18] [20]. According to the Capers, the requirement errors account for 30% defects in MIS applications, 15% in software in the system, 25% in software's of military systems and overall 25% [18][21]. A study done by Ray Rubey shows that, incomplete specification of requirements account for 28% of the defects and intentional deviation from the specified specifications caused 12% of defects [18] [22]. The above data indicates that by clearly specifying the requirements, quality of the software can be improved.

Improvement in the quality also requires the practical implementation of requirements specified. The implementation process involves project planning, project budget (cost and time), and software lifecycle, designing, coding, and testing. Technical documentation and user manual for help are also required. Challenges are faced during the communication between different teams, during interfacing, ensuring cost and time lines, keeping the software bug free, verifying that the software is meeting the requirements, if not, then taking appropriate actions to make the changes.

With the increase in the factors like, frequent change in requirements, shortage of cost and time lines, lack of co-ordination or communication between developers or testers, there will be a chance of building accidental complexity (bugs or errors or unwanted behaviour of the software). The increase in this complexity will result in the decrement of software quality. The Figure 1 gives the relation between the quality and complexity of software. From Figure 1, it can be said that, with the increase in the complexity the quality will decrease and after some point the software will be of no use, as the complexity takes over the quality of software.

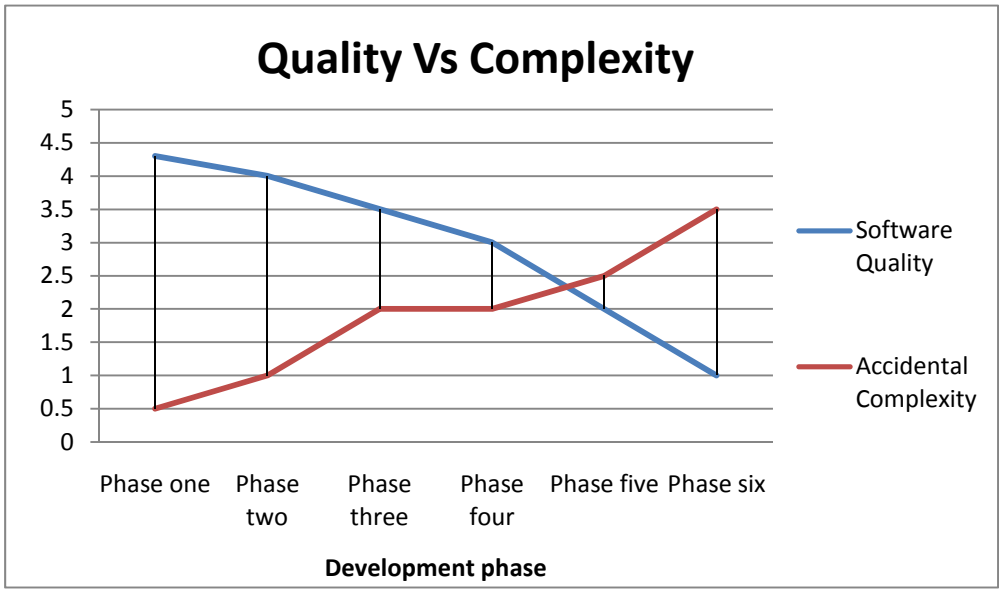

Figure 1: Quality Vs Complexity

The software's developed will always have some complexity in it, but it will be in minimum. Care should be taken such that it continues to remain in the same minimum

level. Organizations developing software must have some methods to know the development process and to keep a track of the software development. Software metrics is one such mechanism which is used to determine the quality of software and keeps track of ongoing project process, software products, and software development process.

There has been a model called "V-Model for Quality"- which gives the principal elements required for the software quality [23]. Its aim is to simplify the complexity involved during the software development. This model contains features that will help the developers to develop software with high quality in an effective manner. First and the most effective use of this model is that, one can easily find the problem without proceeding to implementation and testing. The V-Model for Quality is as shown in figure 2. In this model the quality of each and every phase has to be reviewed before going to the next phase of development.

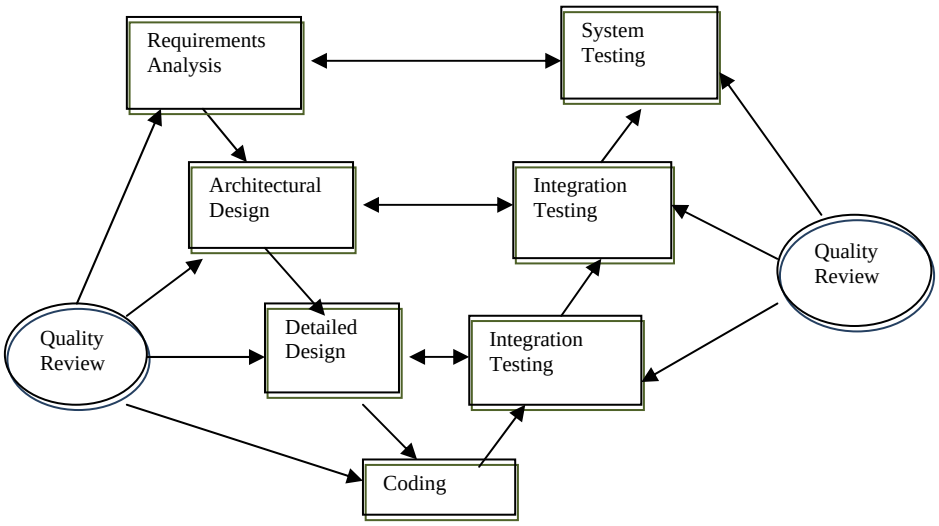

Figure 2: Quality V-Model [23]

In Figure 2, the "quality review" inside the oval indicate the deliverables produced in each and every phase of development i.e., software requirements specification, software design, coding, system testing, integration testing and unit testing. The rectangles on the left side of the diagram signifies the different development phases involved and the rectangles on the right side are the different types of testing for the software. The double headed arrows in the middle connect the different development phases on left and right. The arrow pointing towards the left indicates the dynamic testing of the against the software artifact and the arrows pointing towards right gives the test plans that should be taken place in parallel with the development phase. Although the V-Model for quality provides an outline of what software quality process should contain, it does not provide the details and support needed to implement such a model [23].

3.1 Standards for Software Quality

There are some set of quality standards applicable for the organizations involved in software development. The defined standards must be met by the organization involved in software development [24]. Some standard organizations responsible for giving the standards are [25]:

1. IEEE: Institute of Electrical and Electronics Engineers
2. ISO**:** International Organization for Standardization
3. ANSI**:** American National Standards Institute.
4. AIAA**:** American Institute of Aeronautics and Astronautics.
5. EIA**:** Electronic Industries Association.
6. IEC**:** International Electro technical Commission

Different standard organizations have different definition for their quality standards. ANSI is the only organization that does not make standards but it approves the standards provided [25]. IEEE and ISO are the most widely used standards for computer science. The IEEE has provided the standards of computer software for software quality metrics.

The ISO has provided the standard and framework for information technology, for the evaluation of software quality [ISO: 91]. ISO 9126 defines six product quality characteristics, which has to be met by the software products and the quality of the software's is judged by those characteristics. Those quality characteristics are:

1. Functionality: It indicates the extent to which the functions specified are made available in the software.
2. Reliability: It indicates the reliability of the software.
3. Usability: It indicates the usability i.e., the extent to which the user's feel the software as easy to use.
4. Efficiency: It indicates the efficiency of the software.
5. Maintainability: It indicates the extent to which the software can be easily modified and maintained.
6. Portability: It indicates the ease of changing the software from one environment to another.

The extent of software products exhibiting the above quality characteristics will be the extent of software product quality rating by customers [26]. The organizations developing software products will need some measurement techniques to check the extent to which the product satisfies the above specified characteristic, or the characteristics specified by some other standard. This is when the metrics comes in to existence. Its purpose is to make assessments during and after the software development, so as to know whether the software quality requirements are being met or not.

4. Software Metrics

Software metrics provide measurement for certain aspects of software. The usage of metrics will reduce the subjectivity during the assessment of software quality and it provides quantitative basis for making decisions about the software quality [17] [26]. Metrics can also be used to recognize the duplicated code which can later be removed by applying appropriate refactoring [8] [27] [28]. As we have discussed earlier, software metrics is divided in to two types: software product metrics and software process metrics. Software product metrics is used to measure the final products of the software, for example: software code or design documentation. Software process metrics is used to measure the software development process, for example: Type of methodology and overall development time.

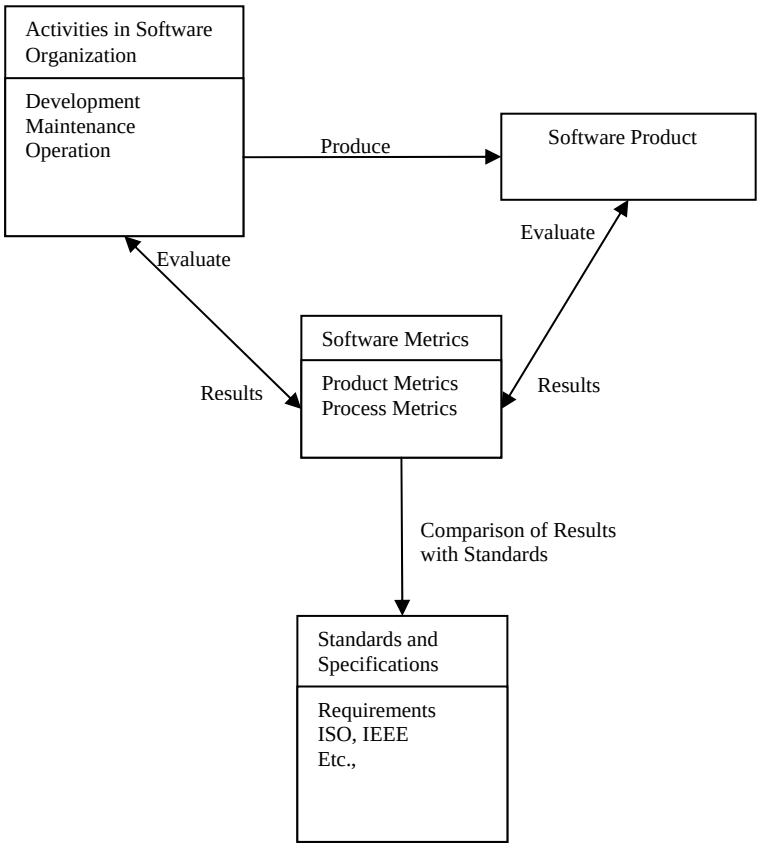

Figure 3: Software Metrics Model

The first level of software metrics begins with the establishment of software quality requirements. All the attributes that define the software quality requirements must be agreed by the management and user-oriented views are then assigned to the attributes [17].Software metrics measuring software products is different for different paradigms. In procedural paradigm, it measures the functions and interaction of functions [30] and in object oriented paradigm, it measures the classes and interaction of classes [29] [30]. In case of procedural paradigm, the function name, type of function, parameters in the function and the interaction between the functions through function calls will make the

structure of the software. Whereas, in object oriented paradigm, class data, operational attributes and the coupling of classes with one another will make the structure of the software.
Figure 3 describes metrics model for selection of metrics and evaluation of metric results. Selection of metrics depends on the development phase of software product. If it is in the starting of development then, process metrics will be used and if the development is in the final phase (i.e., before the customer approval) then, product metrics will be used. The results obtained from the metrics are then compared with the standard or the sections of the software product. The evaluation will be human.

5. Product Metrics

Product metrics are usually derived from the system itself [31]. The metrics data of this type can be collected after specific time intervals. The initial work in product metrics deals with the characteristics of the source code. It is always better to have metric information in the early stages of development because; it will increase the chances of controlling the development process and the results. The following are some examples of metrics which are discussed below:

5.1 Size Metrics

The size of function is regarded as; one of the controversial but still it is the one of the most widely used metrics [29] [30]. It becomes controversial because there is no perfect measure for size, which everyone agrees on. The size metrics is an attempt to quantify the 'size' of software, and the widely used size metrics is Lines of Code (LOC). The size metrics has some deficiencies because it cannot be measured until the process of development is completed. Some Halstead's metrics are also used to measure the size metrics, but they are not discussed in this thesis work.

5.1.1 Lines of Code (LOC)

Lines of Code (LOC) is one of the most widely used metrics for the program size [32]. LOC is calculated by the total number of lines of code in a function. The total number lines can be with or without the blank and comment lines [30]. The decision to include the blank and comment lines will be of the developers. The size metrics can be extended to measure the size of a system by summing all the LOC metric values of all the functions in the system. The calculated values of lines of code metrics is shown in results section (Table 2 to Table 6).

5.2 Complexity Metrics

Complexity metrics is considered as the measure of control flow in a function. The complexity metrics is used to quantify the relation between the complex codes and its failures. The example of Complexity Metrics is Cyclomatic Complexity Metrics.

5.2.1 Cyclomatic Complexity Metrics

Cyclomatic Complexity metrics was proposed by McCabe in the year 1976. It is a measure derived from the product itself [2] [33]. It is used to measure the control flow complexity in a function. It is also considered as one of the internal metrics, as it built

early warning from the collection of the collection of internal metrics [34]. The measured values of cyclomatic complexity metrics can be calculated numerically or can be represented in figures. There are tools for representing the cyclomatic complexity in figures. The calculated cyclomatic complexity is shown in results section (Figure 4, Figure 6 to Figure 9).

5.3 Defect Metrics

It is an external measure of the system derived from the external assessment of the behaviour of the system [33]. It is used to measure the number of defects in a software product and the data required for the metrics is collected from the product itself. So, it can be said that it quantifies the product metrics. There has been no particular procedure for the measurement of number of defects. One of the alternative methods for the measurement of defect metrics is:

- To find the number of errors detected during code inspection.

6. Method

The Method used in this thesis consists of both investigation and practical approach. The investigation is on:

- Software Quality through case study of different journals, text books, research papers, online material and by the usage of standards such as ISO.

- Some of the Software Quality Metrics, such as Product metrics through journals, text books, research papers, online material and by the usage of standards such as IEEE.

The practical approach is on Software Product Metrics, such as:

- Lines of Code: The Lines of Code metrics can be found by using the integrated development environment (for example: eclipse) or by running code in compiler, which gives the total number of lines and in case of any error in the code, it also gives the line of error.

- Cyclomatic Complexity Metrics: It can be found by the using the software Cyvis [37], in which the metrics to find the complexity, total number of methods and the number of lines in each method is predefined.

- Defect Metrics: It is the total number of errors found during the execution of program.

7. Results

Lines of Code:

The lines of code calculated are the total number of executable lines, i.e., excluding the comment lines. The summary of java source code and the summary of each of its classes are given in the tabular columns from Table 2 to Table 6. The results are further discussed in discussions section.

Table 2: Summary of Lines of Code Metrics

Number of Classes	Class Name	Lines of Code	Number of Methods
1	Class 1	329	12
2	Class 2	302	12
3	Class 3	32	2
4	Class 4	18	2

Table 3: Summary of Lines of Code Metrics for Class 1

Number of Methods	Method	Lines of Code
1	Run server	70
2	Process connection	43
3	Send data	38
4	Wait for connection	31
5	Close connection	29
6	Get stream	28
7	<init>	59
8	Display image	9
9	Set text field editable	9
10	Access $000	5
11	Access $100	4
12	Access $200	4

Table 4: Summary of Lines of code Metrics for Class 2

Number of Methods	Method	Lines of code
1	Run client	44
2	Process connection	39
3	Send data	38
4	Connect to server	30
5	Close connection	29
6	Get stream	28
7	<init>	63
8	Display image	9
9	Set text field editable	9
10	Access $000	5
11	Access $100	4
12	Access $200	4

Table 5: Summary of Lines of Code Metrics for Class 3

Number of Methods	Method	Lines of code
1	Main	28
2	<init>	4

Table 6: Summary of Lines of Code Metrics for Class 4

Number of Methods	Method	Lines of code
1	Main	14
2	<init>	4

Cyclomatic Complexity:

The cyclomatic complexity of each class is calculated by counting the number of methods, and the complexity involved during its control flow. The results of cyclomatic complexity are shown in Figure 4, Figure 6 to Figure 9 and it is further discussed in discussions section.

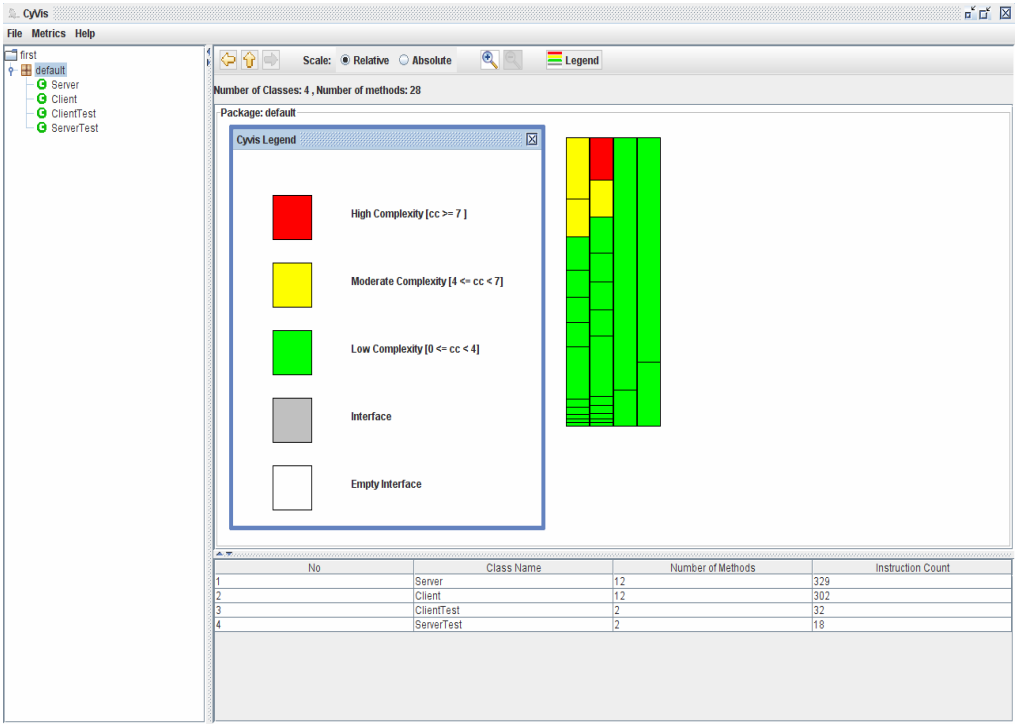

Figure 4: Cyclomatic Complexity

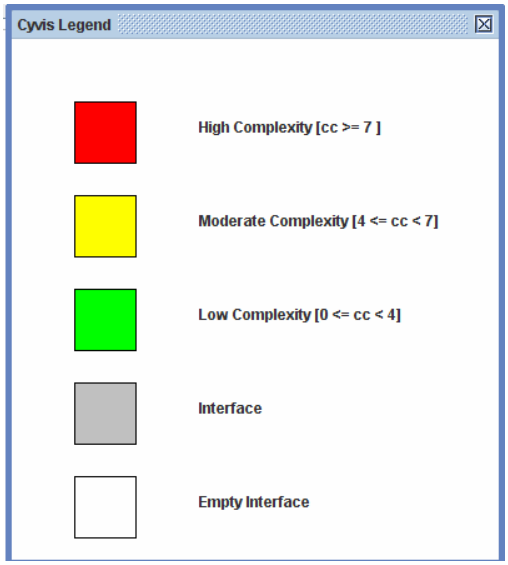

Figure 5: Color Coding for Cyclomatic Complexity Metrics [37]

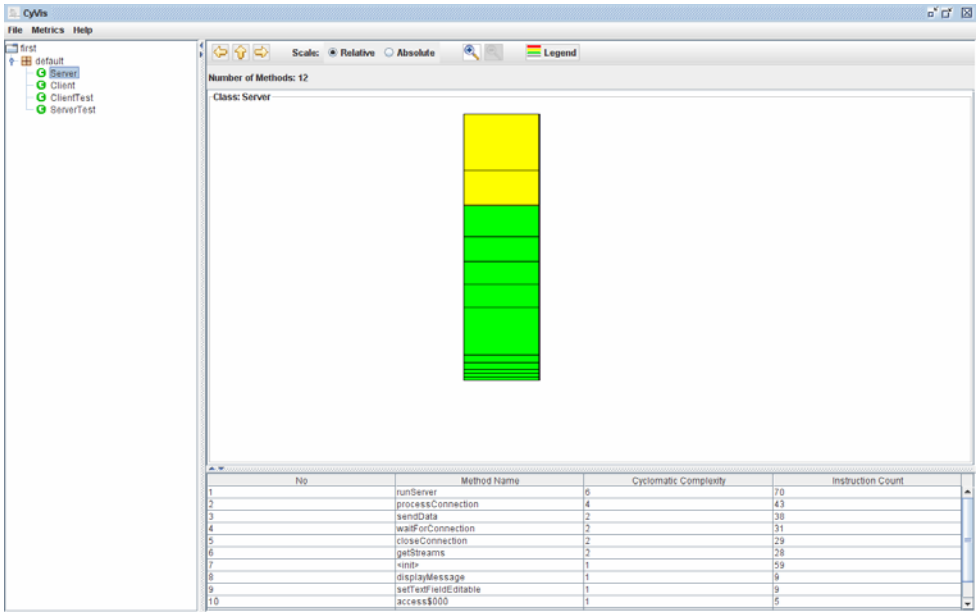

Figure 6: Cyclomatic Complexity for Class 1

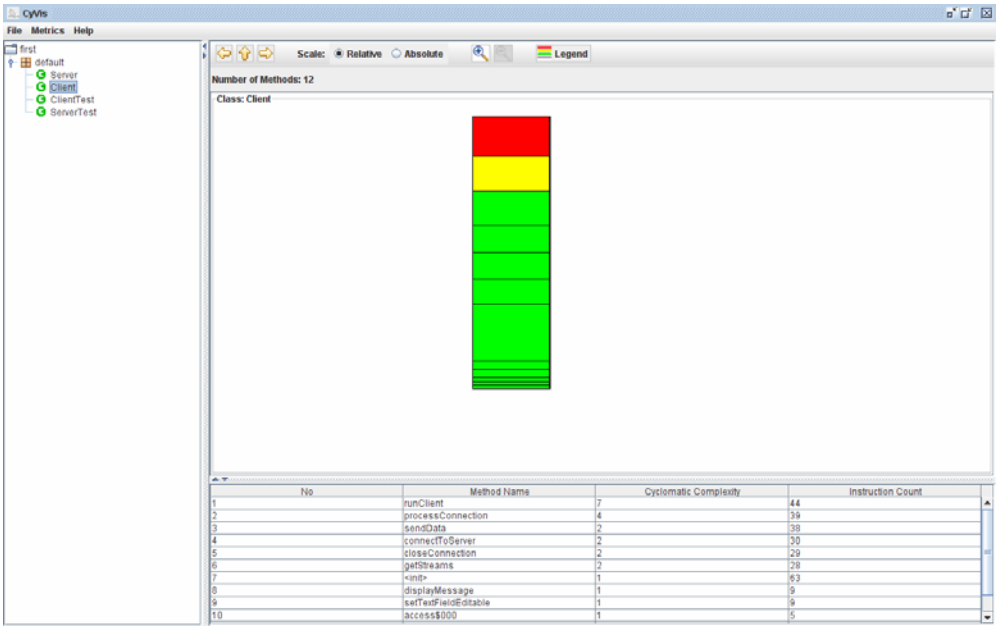

Figure 7: Cyclomatic Complexity for Class 2

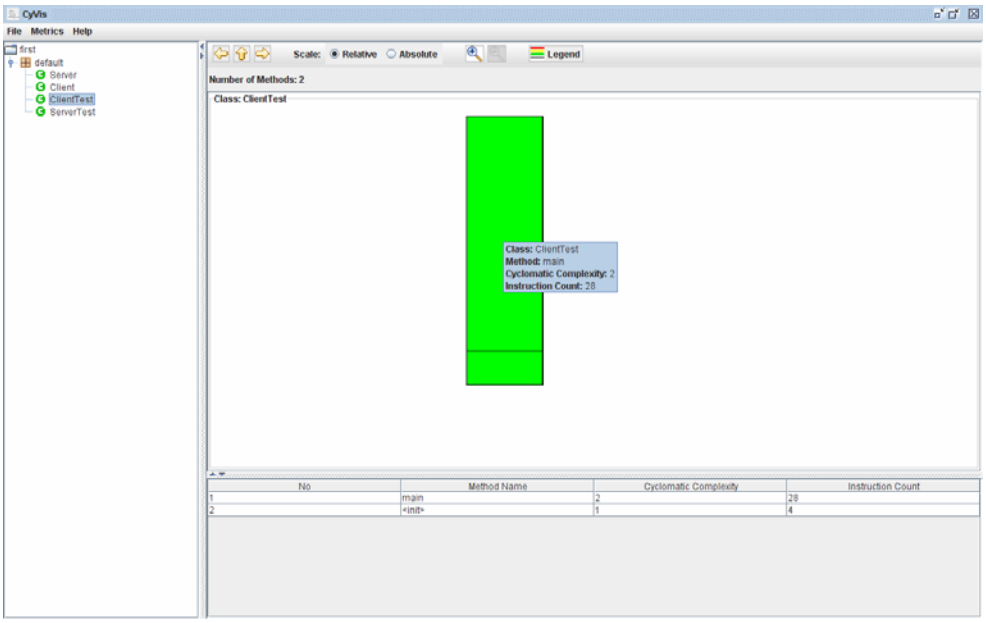

Figure 8: Cyclomatic Complexity for Class 3

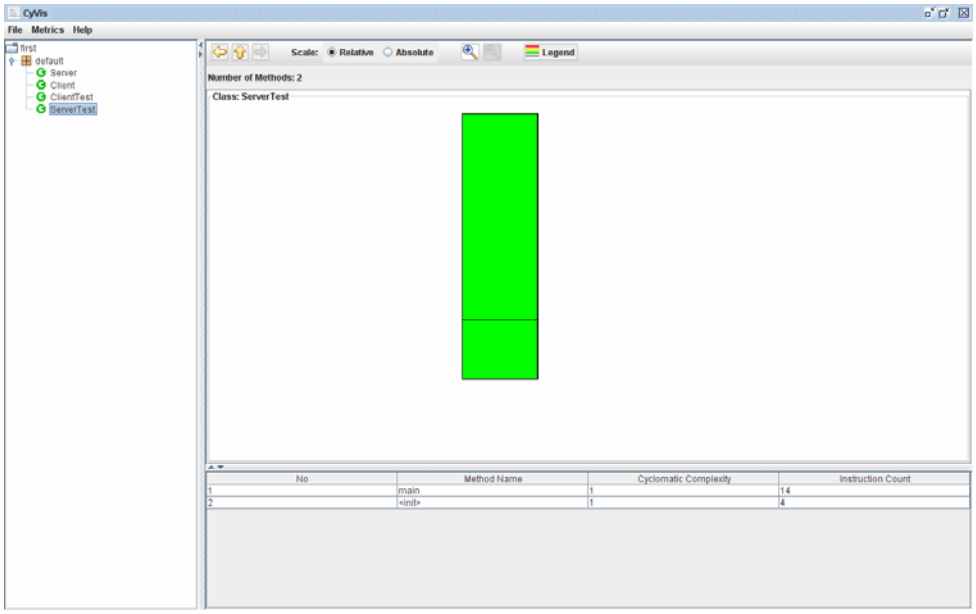

Figure 9: Cyclomatic Complexity for Class 4

Defect Metrics:

The java source code has been inspected and the total number of errors has been found. The result of defect metrics is further discussed in discussions section.

Number of errors detected during code inspection: 32

8. Discussion

The metrics may not directly define quality or can be related to quality. However, they can be used to improve the quality. They can be used to define the parameters that affect the quality and also the changes that can be made to improve the quality. The other main advantages of these metrics are that, they can be used to create the test cases for software testing. They also provide us with the information such as the number of lines in the code, the most complex part of code and also the number of methods contained in the code.

Each of the metrics provides us with specific information of the code. The lines of code metrics represent the size of program and also the number of methods involved in the program. The results of the Lines of Code Metrics are discussed below:

- Table 2 gives the summary of java source code. It consists of 4 classes and the metrics for lines of code for each is calculated. The number of methods involved in each class is also calculated. By summing the total number of lines for each code, the total size of the system can be found.

- Table 3 to table 6 gives the summary for Class 1, class 2, class 3 and class 4. The usage of this metrics will reduce the subjectivity by providing the total

number of lines in each class and the number methods present in it. It makes the software more clear and visible.

- The lines of code for each class can be cross checked by comparing it with summary of classes given in Table 3, Table 4, Table 5, and Table 6.

Cyclomatic complexity, apart from providing us with the complexity in each and every method involved in the code, also provides us with the flow of complexity i.e., structural complexity. It also indicates how complicated the flow is in a function and also indicates how many test cases are needed to perform the basis path testing on the function. The results of Cyclomatic Complexity Metrics are discussed below:

- Figure 4 shows the cyclomatic complexity for java source code. As discussed in lines of code metrics, it consists of 4 classes and flow complexity is shown in Figure 4. The vertical bars represent the classes, and it is from left to right. The horizontal bars represent the method involved in each class, and it is from top to bottom. The colors shaded in each method represent the cyclomatic complexity of that method.

- The meaning of the color and its level of complexity is shown in Figure 5. The method with red color will have the highest cyclomatic complexity, and its value will be greater than or equal to 7. Yellow color represents moderate cyclomatic complexity with its value ranging between 4 and 7. Green color is for low cyclomatic complexity and it will range between 0 and 4. As there has been no interfacing in the java source code, interfacing is not being discussed. But its color representation is shown Figure 5.

- Figure 6 shows the cyclomatic complexity for class 1. From Figure 6, it can be said that the first two methods divided by horizontal lines have the moderate complexity, and the ten methods below it have the low level of complexity.

- Figure 7 shows the cyclomatic complexity for class 2. From Figure 7, it can be said that the first method in this class has the highest complexity, followed by the second method with moderate complexity, and the remaining ten methods have the low complexity.

- Figure 8 shows the cyclomatic complexity for class 3. From Figure 8, it can be said that the two methods in it have the low level of complexity. The value of the complexity of particular method can be viewed at the bottom of Figure 6, Figure 7, Figure 8, and Figure 9. The value of complexity can also be found by placing the arrow over the particular method as shown in Figure 8.

- Figure 9 shows the cyclomatic complexity for the class 4. From Figure 9, it can be said that the two methods in it have the low level of complexity.

Defect metrics does not have particular procedure to measure the total number of defects in the system. The alternative method is to calculate some of the characteristics of the code. As the java source code has been provided after its development, only one characteristic of it has been calculated i.e., the total number of errors during code inspection. The java source code has been inspected and the total number of errors during inspection has been found.

Conclusion

In this thesis the software quality, software metrics and some of applications of software quality metrics has been studied, analyzed and reviewed. The java code has been evaluated using pre defined metrics and the value of different metrics was calculated. From the calculated values of metrics i.e., lines of code, number of errors, and cyclomatic complexity, it was clear that, these metrics can be successfully used to predict the quality level of the software developed.

Lines of code can be used for Maintenance of the program. By having a clear understanding of the number of lines in the code, it will be easy to maintain and also to evolve (Table 2 to Table 6).

Cyclomatic Complexity indicates the part of the code where the complexity is more. Thus more concentration should be given to that part as it is the most error prone part of the software (Figure 4, Figure 6 to Figure 9).

Defect Metrics gives the number of errors that is found during code inspection. So if more than one code inspection is performed, the number of errors found can be compared and it could be checked whether the number of faults decreases with each test.

Lines of code can be used to determine the level of quality by calculating the ratio between the number of errors and number of lines in the code. The lesser the ratio the higher will be the quality.

It is safe to say that the main goals of this thesis were successfully achieved. This thesis could be further expanded by using a code with a higher level of complexity and by using various other metrics and software's. However, the usage of metrics will not eliminate the need for human judgment during the evaluation of software. The usage of software metrics is within an organization and it usage is expected to have a beneficial effect on software organizations by make software quality more visible.

References:

1. Barbara Kitchenham and Shari Lawrence Pfleeger: "Software *Quality: The Exclusive Target*'. IEEE publication, January 1996.

2. Nachiappan Nagappan, Laurie Williams, Mladen Vouk, and Jason Osborne:" *Early Estimation of Software Quality Using In-Process Testing Metrics: A Controlled Case Study"*. Microsoft Research, Redmond, WA 98052, ACM publications

3. Boehm, B. W.: "*Software Engineering Economics"*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

4. J. E. Gaffney, Jr.: "*Metrics in software quality assurance"*. January 1981, ACM 81: Proceedings of the ACM '81 conference.

5. Wei Li and Harry Delugach: "*Software Metrics and Application Domain Complexity"*. Computer Science Department The University of Alabama in Huntsville Huntsville, AL 35899, IEEE publications 1997.

6. B.W. Boehm, J. R. Brown, and M. Lipow: "*Quantitative evaluation of software quality"*. In Proceedings of the 2nd International Conference on Software engineering, pages 592–605, 1976.

7. Konstantinos Stroggylos and Diomidis Spinellis: "*Refactoring – Does it improve software quality?"*. IEEE publications, Fifth International Workshop on Software Quality 2007.

8. Tom Mens and Serge Demeyer: " *Future Trends in Software Evolution Metrics"*. ACM publications 2002.

9. N. Fenton and S. L. Pfleeger: "*Software Metrics: A Rigorous and Practical Approach"*. International Thomson Computer Press, London, UK, second edition, 1997.

10. Demeyer and S. Dueasse. Mettles: "*Do they really help? In Proc. Languages at Modules and Objects"*. Hermes Science Publication, pages 69-82.

11. S. R. Chidamber and C. E Kemerer: "*A metrics suite for Object-oriented design"*. IEEE Trans. Software Engineering, June 1994.

12. Jeffrey Voas: " *A New Generation of Software Quality Conferences*", IEEE publications, IEEE Software January/February 2000.

13. Barry Boehm, Sunita Chulani, June Verner and Bernard Wong: "*Fifth Workshop on Software Quality*". IEEE Publications, 29th International Conference on Software Engineering

14. Kitchenham, B: " *The Failure of Quality, Proceedings of the Second Workshop on Software Quality". ICSE 2004.*

15. "*Software Metrics - An Introduction*". IEEE publications.

16. Konstantinos Stroggylos, Diomidis Spinellis: "*Refactoring – Does it improve software quality?*". IEEE publications, Fifth International Workshop on Software Quality 2007.

17. "*IEEE Standard for a Software Quality Metrics Methodology*". IEEE publications.

18. Evans, Isabel: "*Achieving Software Quality Through Teamwork"*.Norwood, MA, USA: Artech House, Incorporated, 2004.

http://site.ebrary.com/lib/bthbib/Doc?id=10081986&ppg=25

19. Kenett and Ron: "*Software Process Quality: Management and Control"*.New York, NY, USA: Marcel Dekker Incorporated, 1999.

http://site.ebrary.com/lib/bthbib/Doc?id=10051404&ppg=1

20 . Schulmeyer, G. Gordon: "*Software Quality Assurance- Coming to Terms, in The Handbook of Software Quality Assurance"*. Schulmeyer, G. Gordon and McManus, James I., eds., New York: Van Nostrand Reinhold Company, Inc., 2nd ed., 1992.

21. Jones, T. Capers: "*Applied Software Measurement*". McGrawHill, 1991.

22. McCabe, Thomas J., and Schulmeyer, G. Gordon: "*The Pareto Principle Applied to Software Quality Assurance, in The Handbook of Software Quality Assurance*", Schulmeyer, G. Gordon and McManus, James I., eds., New York: Van Nostrand Reinhold Company, Inc., 2nd ed., 1992.

23. Thomas B. Hilburn and Massood Towhidnejad: "*Software Quality Across The Curriculum"*. Published in 32nd ASEE/IEEE Frontiers in Education conference.

24. Hector Morrison: *"Standards and Certification*". IEEE publication 1993.

25. Dr. James A. Bednar and Dr. David Robertson: "*Software Quality and Standards*". SEOC2 Spring 2005, Quality/Standards.

26. O'Regan and Gerard: "*Practical Approach to Software Quality"*. Secaucus, NJ, USA: Springer-Verlag New York, Incorporated, 2002.

http://site.ebrary.com/lib/bthbib/Doc?id=10047715&ppg=19

26. Professor Norman F. Schneidewind: "*R e p o r t on the IEEE Standard for a Software Quality Metrics M e t h o d o l o g y (Draft) P1061, with Discussion of Metrics Validation*', IEEE publications 1991.

27. K. Kontogiannis: "*Evaluation experiments on the detection of programming patterns using software metrics"*. IEEE Computer Society Press, 1997.

28. B. Lagufi, D. Proulx, E. M. Merlo, J. Mayrand, and J. Hudepohl: "*Assessing the benefits of incorporating function clone detection in a development process"*. IEEE Computer Society Press, 1997.

29. Wei Li and Sallie Henry: "*Maintenance Metrics for the Object Oriented Paradigm*". IEEE publications

30. Wei li: " *Software Product Metrics – Using them to Quantify Design and Code Quality*". IEEE publications, December 1999/ January 2000.

31. Nachiappan Nagappan, Thomas Ball, and Brendan Murphy: "*Using Historical In-Process and Product Metrics for Early Estimation of Software Failures*" Microsoft Research, IEEE publications.

32. Everald E. Mills: "*Software Metrics*'. SEI Curriculum Module SEI-CM-12-1.1, December 1988.

33. ISO/IEC: "*DIS 14598-1 Information Technology – Software Product Evaluation*". ISO 1996.

34. McCabe, T. J.: "*A Complexity Measure*". IEEE Transactions on Software Engineering, Vol. 2, No. 4, pp. 308-320, 1976.

35. Chulani, S, Ray, B., Santhanam, P. and Leszkowicz, R.:"*Metrics for Managing Customer View of Quality*", IEEE Metrics conference, Sep. 2003.

36. M. Lorenz and J. Kidd: "*Object-Oriented Software Metrics: A Practical Approach"*. IEEE publications.

37. http://cyvis.sourceforge.net/