ASSESSMENT AND EVALUATION OF QUALITY ASPECTS OF SOFTWARE SYSTEMS USING MULTI CRITERIA DECISION APPROACH

THESIS

submitted in fulfilment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

to

J.C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA

by

AMANDEEP KAUR

Registration No: YMCAUST/PH51/2011

under the supervision of

Dr. ASHUTOSH DIXIT
PROFESSOR

Dr. P.S. GROVER
PROFESSOR
& FORMER DEAN (DU)

Department of Computer Engineering
Faculty of Engineering and Technology
J.C. Bose University of Science &Technology, YMCA
Sector-6, Mathura Road, Faridabad, Haryana, INDIA
September, 2019

DECLARATION

I hereby declare that this thesis entitled "**ASSESSMENT AND EVALUATION OF QUALITY ASPECTS OF SOFTWARE SYSTEMS USING MULTI CRITERIA DECISION APPROACH"** by **AMANDEEP KAUR,** being submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy in Department of Computer Engineering under Faculty of Informatics and Computing of J.C Bose University of Science and Technology, YMCA, Faridabad, during the academic year 2019-2020, is a bona-fide record of my original work carried out under the guidance and supervision of **Dr ASHUTOSH DIXIT, PROFESSOR, DEPARTMENT OF COMPUTER ENGINEERING, J.C BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY, YMCA, FARIDABAD** and **Dr P.S. GROVER, PROFESSOR AND FORMER HEAD/DEAN , DEPARTMENT OF COMPUTER SCIENCE, DELHI UNIVERSITY, DELHI** and has not been presented elsewhere.

I further declare that the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

(AMANDEEP KAUR)

Registration No: YMCAUST/PH51/2011

CERTIFICATE

This is to certify that this thesis entitled "**ASSESSMENT AND EVALUATION OF QUALITY ASPECTS OF SOFTWARE SYSTEMS USING MULTI CRITERIA DECISION APPROACH"** by **AMANDEEP KAUR,** submitted in fulfillment of the requirement for the award of Degree of Doctor of Philosophy in Department of Computer Engineering, under Faculty of Informatics and Computing of J.C Bose University of Science and Technology, YMCA, Faridabad, during the academic year 2019-2020, is a bona-fide record of work carried out under our guidance and supervision.

We further declare that to the best of our knowledge, the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

DR. ASHUTOSH DIXIT
Professor,
Department of Computer Engineering,
Faculty of Informatics and Computing,
J.C Bose University of Science and Technology,
YMCA,
Faridabad

DR. P.S. GROVER
Professor and Former Dean/Head,
Department of Computer Science,
Delhi University,
Delhi

Date:

ACKNOWLEDGEMENT

I express my gratitude to almighty WAHEGURU JI for giving me strength, positivity and courage to complete this thesis.

I would like to express my sincere and deep gratitude to my supervisors, **Dr Ashutosh Dixit** Professor, Department of Computer Engineering, J.C.Bose University of Science & Technology, YMCA, Faridabad and **Dr P.S. Grover,** Professor and Former Head/Dean, Computer Science Department, Delhi University, Delhi for giving me the opportunity to work in this area. It would never be possible for me to take this thesis to this level without their continuous guidance, innovative ideas, constructive criticism and valuable advice. Their knowledge in various perspectives of research provided me the opportunity to make significant work in this direction. Their continuous support and encouragement keep me sailing through tough times of my Ph.D. pursuit.

I am sincerely thankful to Dr Komal Bhatia, Chairman, Department of Computer Engineering and Dr Atul Mishra, Chairman, Department of Information Technology Computer Application for their constant feedback and suggestions followed by consistent encouragement during the course of this research work. I want to express my special thanks to Dr Naresh Chauhan for his motivation and moral support for doing research in better way. I would like to thank all the faculty members of Department of Computer Engineering for their constant cooperation and all time support.

A special thanks to my grandmother Late Sdr. Mohinder Kaur for showering me with unconditional love and encouragement throughout my education. I am sincerely thankful to my parents S. Jagdev Singh and Sdr. Kamaljeet Kaur, and my brother S. Japneet Singh for being there for me always and showering me their blessings and support. I am also thankful to my brother-in-law S. Gurpreet Singh for his continuous availability and support.

No words are sufficient to thank my better half, S. Tajinder Singh and my sweethearts, Baby Gurnoor Kaur and Master Harvansh Singh for their understanding with utmost patience, love and faithful support without which it would never be possible to finish this thesis.

Thank you all**!**

(Amandeep Kaur)

DEDICATED

to

“MY LOVING FAMILY”

for their constant support with endless patience.

ABSTRACT

IT world is evolving rapidly and the software development companies are trying to carve out their places in it. Everything is getting automated and is getting software based. With this the dependency on the software is increasing rapidly. Hence, the responsibility of developing quality software has enhanced. Constructing high quality software is a huge challenge. Traditionally the focus of software industry was to provide more and more functionality in a software product. But over the last decade this focus has been shifted on to improving quality of the software product. Indisputably the software quality has become an indispensable subject in the field of software engineering. Eventually, developing trustworthy software is the new focus of software developers along with the functional software. While developing a project, developer has to deal with a number of constraints related to functionality, quality, cost and time. In order to meet all the constraints, new software development methodologies keep on flooding. The range of software development approaches can be pigeonholed into three broad genres. First, the modular programming approach, which was introduced in 70's, which is then followed by object oriented programming approach in the early 90's and thirdly aspect oriented programming approach that is evolved on the concept of OOP approach in the later years of the 19th century.

Aspect oriented software development approach is relatively new approach for software development that is gaining attention. AOP methodology proposes to remove the cross-cutting and tangling problems of OOP approach by the proper implementation of the concept called as separation of concerns. Separation of concerns is an important concept and activity in structuring the software systems meaningfully and can play an important role in adding the trustworthiness to the software. Aspect-oriented programming promises to enhance the extensibility and reusability of code through the removal of tangled and crosscutting code by the usages of aspects and pointcuts.

Moreover as every now and then new software programming techniques are being proposed. In order to assess, evaluate and improvise the software quality, different

software quality models have been proposed by various researchers as well as consortiums. Software metrics are a way to quantify qualitative attributes of the software. Hence, to measure the numerous attributes of the software, various researchers have proposed various software metrics. But AOP being a new paradigm relatively, therefore, a lesser number of software metrics are proposed and validated till date. To initiate the research process, Aspect Oriented Metrics proposed by the researchers have been studied, analyzed and it's affect on the trustworthiness and hence on quality of the aspect oriented software is tabulated.

So, thrust of the thesis is to assess and evaluate the relevant quality aspects of the software systems by developing a software product quality model that could incorporate the modern-day software features.

Towards this goal, firstly various old and new software quality models are reviewed critically and their weaknesses identified. Further, various missing but relevant characteristics are identified on the basis of latest ISO/IEC software quality model ISO 25010. Four characteristics are found to be essential to be part of the proposed software quality model. Finally an improved software product quality model is proposed to incorporate the relevant modern-day software features. The validation of the proposed model is done using Analytical Hierarchy Process (AHP) technique of Multi-Criteria Decision Method (MCDM).

Further, the reusability and complexity of aspect oriented systems is studied and compared it with object oriented Systems using OO metrics. Through statistical analysis it is found that aspect oriented systems are better as regards stability, reusability and maintainability though at the cost of higher complexity.

The work has also put forward three unified frameworks for the quality attribute evaluation.

Firstly, considering that determining coupling for Aspect-oriented Systems (AOSs) would assist in the quantification of various software attributes and hence improve quality. This paper presents a new Aspect-oriented System Coupling Metric (COAO), which is based on the properties of elements and the relationships among them. The

process of defining a metric primarily requires a clear, unambiguous definition of basic and relevant concepts related to Aspect-Oriented Programming. As such, first and foremost, novel definitions of basic concepts such as system, element, relation, module and attribute are specified concerning Aspect-Oriented Programming. Subsequently, a current metric for Aspect-Oriented System Coupling is proposed, followed by an illustration of exemplary calculation for the proposed metric. Finally, the proposed metric is validated theoretically against Braiand properties for coupling of software systems. The result indicates that the proposed metric for the Aspect-oriented System Coupling metric COAO is a valid metric for measuring coupling in Aspect-oriented Software Systems.

Secondly, as it is considered that the computation process inclusive of the systematic approach leads to fairly good solution with a high degree of consensus. Hence a metric is proposed for the assessment of supportability attribute with the aid of an exemplary real-life questionnaire. Also, a case study has been performed for the evaluation of Supportability metric.

Thirdly, it is analyzed that there is a need for a formal framework for evaluating the extensibility of the software. In order to design a framework for extensibility, a novel maintainability model (proposed and validated earlier) for aspect-oriented software systems on the lines of ISO/IEC 25010 has been used. By applying the maintainability model, a novel framework for evaluating the extensibility characteristic is exhibited. The proposed framework is tested for a set of aspect-based software. Also, validation of the proposed extensibility metric is done by applying Karl Pearson Product Moment Correlation method. Finally, a comparison is made between software built using object-oriented approach and aspect-oriented approach. Results suggest that software built using aspect-oriented approach is more extensible than the one built using object-oriented approach. The proposed framework is found to help software developers in selecting software that can be easily extensible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AOP	Aspect-Oriented Programming
MOP	Modular Programming
OOP	Object-Oriented Programming
AOSD	Aspect-Oriented Software Development
ITD	Introduction
IoT	Internet of Things
AHP	Analytical Hierarchical Process
MCDM	Multi Criteria Decision Making
LOC	Lines Of Code
CK	Chidamber and Kamerer
WMC	Weighted Methods per Class
RFC	Response For a Class
LCOM	Lack of Cohesion of Methods
CBO	Coupling between object classes
DIT	Depth of Inheritance Tree
NOC	Number of Children
LK	Lorenz and Kidd
CS	Class Size
NOO	Number Of Operations
NOA	Number Of Added operations
SI	Specialization Index
MOOD	Metrics for Object-Oriented Design
MHF	Method Hiding Factor

AHF	Attribute Hiding Factor
COF	Coupling Factor
MIF	Method Inheritance Factor
AIF	Attribute Inheritance Factor
PF	Polymorphism Factor
CAE	Coupling on Advice Execution
CIM	Coupling on Intercepted Module
CMC	Coupling on Method Calls
CFA	Coupling on Field Access
RFM	Response For a Module
LCO	Lack of Cohesion in Operations
CDA	Crosscutting degree of an Aspect
WOM	Weighted Operations in Module
DIT	Depth of Inheritance Tree
NOC	Number of Children
NOF	Number Of Features
NOA	Number Of Aspects
NCI	Number of Classes and Interfaces
BCF	Base Code Fraction
ACF	Aspects Code Fraction
AF	Advice Fraction
ACD	Advice Crosscutting Degree
PHQ	Program Homogeneity Quotient
CoAT	Coupling on Attribute Type

CoPT	Coupling on Parameter Type
CoAR	Coupling on Attribute Reference
CoOI	Coupling on Operation Invocation
CoI	Coupling on Inheritance
CoHA	Coupling on High Level Association
UACoh	Unified Aspect Cohesion
CMPX	complexity metric of the aspect-oriented system
FURPS	Functionality, Usability, Reliability, Performance and Supportability
DEQUALITE	Design Enhanced Quality Evaluation
AOSQUAMO	Aspect-Oriented Software Quality Model
WSM	Weighted Summation Method
WPM	Weighted Product Method
ANP	Analytical Network Process
ISM	Interpretive Structural Modelling
TOPSIS	Technique for Order Preference by Similarity to Ideal Solution

CHAPTER I

INTRODUCTION

1.1 GENERAL

Software is progressively becoming an essential element of business systems, products, and services. With the significant increase in the production of the software, the necessity of quality has become apparent. It has become critical to assure the availability and dependability of the software. The significant issues in developing modern-day software are the ever-increasing size and complexity of the systems incorporated all together along with the high demands for quality. Quality of software products has evolved to be a key component in the success of any business. Reduced quality of the software product in delicate frameworks like responsive systems, control systems may possibly result in financial loss, permanent damage, mission breakdown, or even loss of human life. During software product development, the developers have to deal with several different, and most of the time competing, aspects of quality requirements that are related to functionality, quality, cost, and time. Software failures cost companies a significant amount of money and cause damage to their brand value, which in turn results in loss of jobs. Poor performance or a lack of functionality within internal and external applications can significantly impair the organization's ability to compete and respond to business demands in a competitive market.

Improper application of the quality in each area of business applications results in expenditure overruns, schedule overruns in addition to the production of wastes in the way of rework up to nearly half of the total development time. Quality build-in software tends to have fewer defects that saves a tremendous amount of time as well as money. Time gets saved through testing and maintenance phases, and maintenance costs get lowered because of the increased reliability of the quality software that contributes appreciably towards client satisfaction; thereby leading to the lower overall expenditure.

Over time, the basic definition of software quality has evolved. As per IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990, term '*Software Quality*' is defined as "the extent to which a system, component or process meets specified requirements and customer needs" [1][2]. This definition was redefined in the year 2010 [113], as "the capability of a software product to satisfy needs, both which are explicitly stated as well as implied, when used under specified conditions"[113]. The ISO/IEC/IEEE 24765; International Standard in the year 2017 superseded the previous ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary and defined software Quality [114] as:-

- "The capability of the software product to satisfy stated and implied needs when used under specified conditions,
- The degree to which a software product satisfies stated and implied needs when used under specified conditions,
- The extent to which a software product meets established requirements."

This description of software quality envelops together the objective as well as the subjective element of quality. However, one thought-provoking question that arises is: 'Who is the Customer?' The first thing that comes into the mind is *External Customers*. That is, those people who are external to the organization and who receive the end product (software) and services. Another category of customers who are often forgotten or taken for granted is *Internal Customers*. These internal customers are the personnel in the next phase of the software development lifecycle and are the takers of the work done during the current phase of the software development [3].

It is quite apparent that quality cannot be added later into the system as an afterthought. Instead, it needs to get built into the system from the very beginning. For developing quality software, there is a need to create an efficient system not only in terms of time and resources, but the efficiency of the code should also be a parameter. That means consideration should be given to the expectations of both external customers as well as internal customers. Besides, these days, reuse of the code is quite frequent and plays a significant role in the design and development of software systems as well [3].

Numerous techniques are proposed by renowned researchers that are required to quantify the quality of the software as quantification plays a vital role in its implementation. However, there cannot be one single way to measure quality in a variety of software products as they are developed using different platforms for numerous applications. In the absence of a single measure, there is a disparity between measuring software quality for different types of software. Hence there are variations in measuring and improving software quality in an embedded system (where the prominence is on risk management), in business software (with emphasis on cost and maintainability management) and in mobile computing (where importance is on user-centric software application on the smartphone depends on the quality of software across all types of software layers).

To facilitate improvement in business performance and to ensure quality maintenance; the developers and researchers adopt the latest software development approaches [2][4]. The subsequent section 1.2 briefs the broad categories of the types of programming.

1.2 TYPES OF PROGRAMMING

Today's commercial environment is changing faster than at any time in history. With the purpose to meet all the types of requirement, various latest software development methodologies keep on evolving with time. Standard software development methodologies mainly comprise of module-oriented, object-oriented, and aspect-oriented programming (AOP) methodology. Earlier the Structured programming technique, also known as modular programming, was introduced. *Modular Programming(MOP)* [2][4] is a technique of designing a software that is established on the concept of modules. Each module has a well-defined interface; purposely designed to accomplish typically only one function and encloses everything essential to achieve it. This programming technique amplifies the degree to which software is organized into separate, interchangeable components by breaking down larger program functions into smaller modules. After modular programming, *Object-Oriented Programming (OOP)* technique [2][4] came into the picture. Concept of classes and objects forms the basis for object-oriented programming paradigm to design computer programs and applications. Precisely objects are the data structures

that consist of attributes (data fields and methods) together with their interactions. OOP technique includes features like data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance.

Aspect-Oriented Software Development (AOSD) [5] is the modern programming paradigm and is being deployed to design and develop software systems. AOSD is a promising technology of software development that seeks new modularizations of software systems intending to segregate secondary or supporting functions from the program's primary business logic [6][7]. Quality assessment of such software systems has been an issue of great importance. Therefore the AOSD approach has been exercised in the thesis to assess the quality aspects of software development and management.

Following Section 1.3 briefs the introduction of Aspect-Oriented Software Development.

1.3 ASPECT-ORIENTED SOFTWARE DEVELOPMENT

Aspects - a new type of abstraction, forms the underlying basis of aspect-oriented software development. In general, as soon the scalability and complexity of software product increases, the object-oriented systems tend to suffer from two pertinent problems: scattered and tangled code. These problems happen for the reason that an object-oriented software product decomposes software along one dimension only [4][5][8]. The fundamental idea behind Aspect-oriented software development is the principle of *separation of concern* [5][6]. The principle of separation of concern states that software ought to be organized in such a manner that every program element does one thing and one thing only. By integrating the separation of concerns in software, there is clear traceability from requirements to implementation. Below are described, briefly, the salient features of AOSD.

1.3.1 Separation of Concerns

Concerns are the features of the software that describe its universal and variable functionalities. Concerns are multidimensional by nature. Separation of concerns is a

key concept and activity while structuring the software systems meaningfully. Broadly there are two categories of concerns.

- *Core concerns*
- *Secondary concerns*

Core concerns are the functional concerns that relate to the primary purpose of a system. While secondary concerns are the concerns that relate to the non-functional and Quality of Service requirements of a system. These types of non-functional concerns spread over various non-related classes; hence, also known as crosscutting concerns [5][6][7]. Logging, security, performance monitoring, transaction management are a few types of concerns that are of cross-cutting nature as they are of the system-level.

Cross-cutting concerns cause problems with both, maintainability as well as efficiency, of the software. For instance, if one of these cross-cutting concerns needs a change at some point in the program's lifetime, multiple modules need to be modified. Cross-cutting concern wreaks havoc on the maintainability of code if not appropriately handled as they cannot be constrained easily into modular form. These concerns tend to hamper the modularity. Implementation of cross-cutting concerns introduces related or even duplicated code into one or more modules that leads to the scattering and tangling of the code. Scattered and tangled code decreases the cohesiveness and the modularity, which in turn increases the complexity of the software. As the complexity of the code increases, the number of defects may also tend to grow. This increase eventually indicates a decrease in the quality of software [9][10][11].

Crosscutting concerns can be broadly categorized into two types:

- *static crosscutting*
- *dynamic crosscutting*

Static crosscutting can alter the static structure of other elements in a program. That means it has the power to add/modify the new members in the primary abstraction or

convert checked exceptions into unchecked exceptions. They are implemented using Introduction feature in AspectJ.

Dynamic crosscutting can change the behavior of the primary abstraction. Typical examples of crosscutting concerns are security, logging, etc. These types of crosscutting concerns are not required to be implemented separately/individually in core functionality.

AOSD provides an organized and systematic means to modularize cross-cutting concerns. AOP offers explicit support for program modularity instead of spreading the code associated with a behavioral requirement or concern throughout a program. Various aspects related to crosscutting concerns implementation can be created and maybe weaved in the code as and when required to execute.

Aspect-oriented program is principally composed of two parts, namely

- *base code*
- *aspect code*

The base code encapsulates the primary functional concerns, and classes are used for their implementation. The aspect code encapsulates the functionality that cross-cuts and co-exists with the other functionality and is implemented by the use of aspects. The aspects include the code implementing the cross-cutting concern (advice) along with the place where it should be executed in a program (pointcut). The aspect-oriented code eventually transformed into object-oriented code/base code with the aspects integrated into the code by the aspect weaver [5]. Aspects are designed to implement crosscutting concern not only easily maintainable but more reusable. Both the base code and aspect code must run in a well-coordinated manner to achieve the objective.

The overview of the Aspect-Oriented Programming Methodology is composed of six foremost concepts [5][6] as depicted in the following Figure 1.1.

- Aspect
- Joinpoint

- Pointcut
- Advice
- Introduction (ITD)
- Weaving

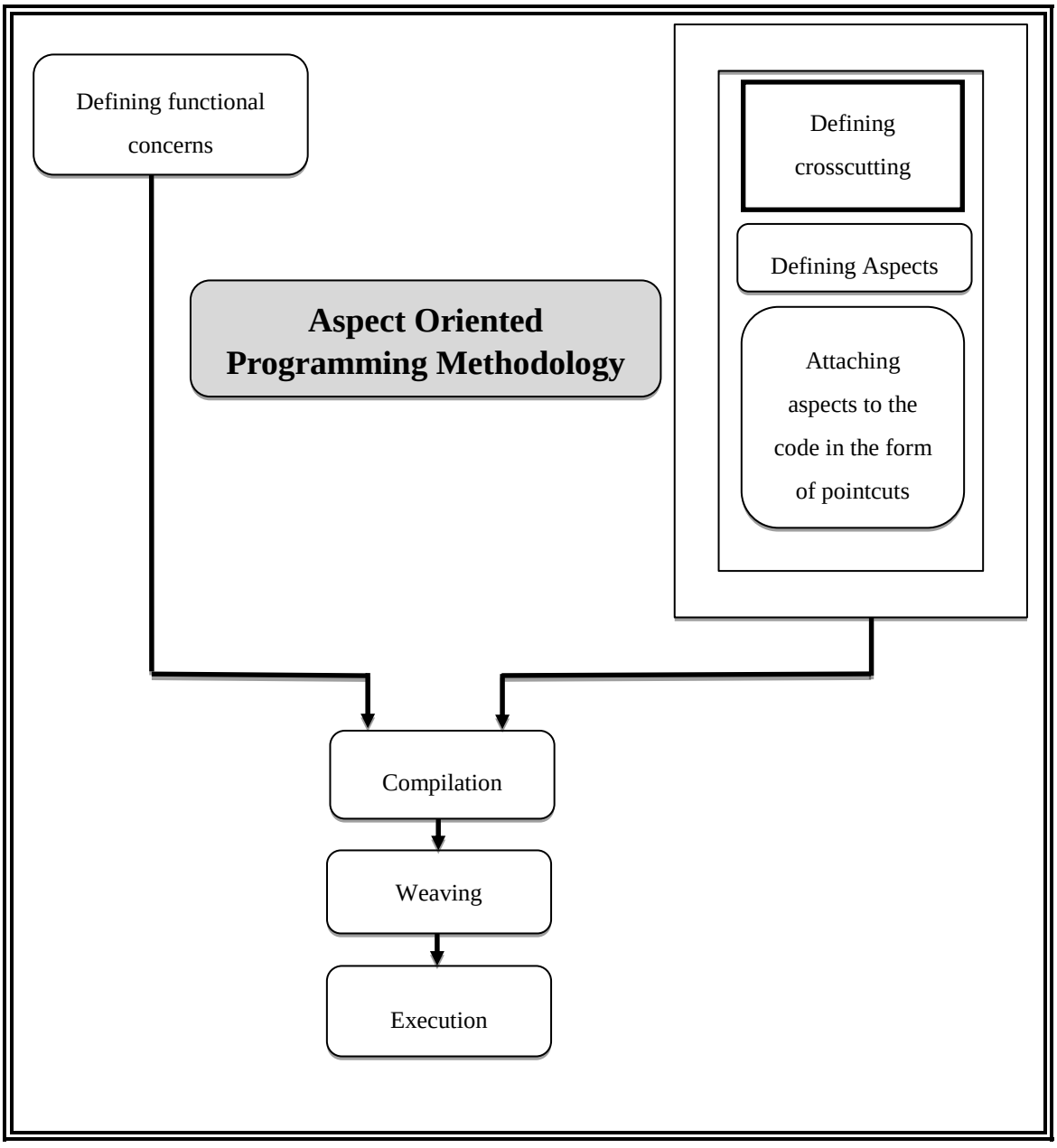


Figure 1.1: Aspect-Oriented Programming Methodology

The primary concepts and terminologies introduced by Aspect-oriented software [5][6] development are summarized as follows:

A. **Aspect:** It is a modular unit for the representation of crosscutting concern that defines when, where, and how of crosscutting concern's invocation and implementation. It is the basic unit of modularization for the implementation of crosscutting concerns.

B. **Joinpoint:** It specifies the well-defined location in the code at which cross-cutting concern needs to be called.

C. **Pointcut:** On the basis of specific criteria, a set of joinpoints in the base code is chosen by the pointcut where the crosscutting code needs to be woven.

D. **Advice:** It is a construct similar to the method used to define cross-cutting code. It is the behavior executed before, after, or around the particular join point when reached during the execution of the base code.

E. **Introduction (ITD):** It is used to add new variables or methods to a class and notifies if the class implements an interface.

F. **Weaving:** It weaves the appropriate advice at the appropriate join point. Figure 1.2 displays the basic functioning of aspect weaving in AOP software. If the code is woven at compile-time, then it is called *Static aspect weaving*. If the code is woven at run time, then it is called *Dynamic aspect weaving* [6].

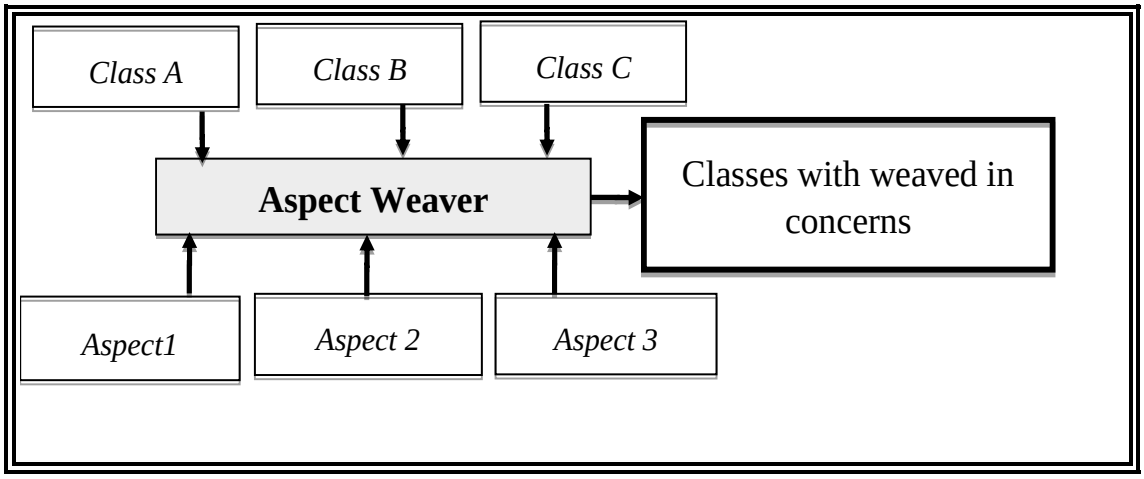


Figure 1.2: Aspect Weaving in Aspect-Oriented Programming

Aspect-oriented programming can significantly reduce the code size of an application by eliminating scattering and tangling in the code. Aspect-oriented programming is expected to have a positive effect on performance, code size, modularity, and evolution [9]. Aspect-oriented programming does not replace object-oriented programming; rather, it enhances its ability to implement crosscutting concerns in a clean, modular way by the use of aspects [1][2].

Aspect-oriented programming is implemented as an extension to other programming languages like AspectC++, AspectJ or by extending libraries like SpringAOP. AspectJ has become the de facto standard for the Aspect-oriented programming languages. It is a simple and practical advancement of the Java programming language that permits the usage of aspect-orientation techniques. AspectJ has a separate set of terminology for modular implementation of crosscutting concerns along with the terminology used in Java. Crosscutting concerns are implemented using Joinpoint, Pointcut, Advice, and Weaving feature of AspectJ.

- **AspectJ**

AspectJ [111] is the most established and predominantly used representative of the Aspect-oriented programming languages. The primary intention of AspectJ is to provide a standardized mechanism to software developers for modularization of a non-functional and cross-cutting concern that does not fit cleanly into a class-object model. It is an aspect-oriented extension to an object-oriented language, Java. It provides a separate construct for programming code called advice that is weaved into the functional core at specific locations called join points. The advice code can be fabricated either before or after these join points. It also provides the pointcut construct for indicating the joinpoints where advice can be executed [6].

Since aspect-oriented and object-oriented software programming paradigms are broadly centered on the separation of concerns related to primary and secondary concerns; so a comparison of the two approaches is given in the next section. Also, the application areas of aspect-oriented approach are explored alongside.

1.3.2 Aspect-Oriented vs Object-Oriented Systems

Currently, object-oriented software systems and aspect-oriented software systems are the paradigms that are accessible and most potent in the software industry [5]. Object-oriented software development is a software development which is based on the objects. Objects are the real-world entities that are represented using classes consisting of data fields and methods together along with their interactions. AOP's idea of the separation of concern ensures modularity. OOP enhances vertical relationships in the form of inheritance but not horizontal relationships. In a well-implemented object-oriented program, functionalities are spread all over the classes in the system.

Traditionally the focus of the software industry was to provide more and more functionality (core concern) in a software product. Over the last decade, the attention of the software industry has shifted from delivering more functionality towards improving quality as perceived by the end-users. As the focus shifted on to improving the quality of the software product, the need for non-functional requirements (secondary concern) has become mandatory. These concerns are generally crosscutting and their implementation, lead to tangled and scattered code. Such code leads to a lot of redundancy, which is not only difficult to maintain but to enhance too.

The manner in which Aspect-oriented approach handles cross-cutting concerns also ensures quality in the final code. Cross-cutting concerns cause problems with both maintainability and efficiency. If they are not handled properly, they make maintainability very difficult. If one of these crosscutting concerns needs to be changed at some point in the program's lifetime, multiple modules need to be modified. Moreover, cross-cutting concerns cannot be constrained easily into a modular form. These type of concerns tend to destroy the modularity of the software. The issues related to duplicate scattered and tangled code; decreased cohesiveness and the modularity; increased software complexity; hence the decreased overall quality of software; have been addressed in the aspect-oriented programming paradigm approach [9].

❖ **Applicability of AOP Approach in Modern Systems**

Modern systems often require connecting existing systems that are distributed physically through the usage of internet. Extensive and heterogeneous data of the Internet of Storage has posed enormous storage challenges which have resulted in making storage a significant research trend of the data management of the Internet of Things (IoT). Data being one of the most ethical aspects of IoT are collected from different classifications of sensors and embodies billions of objects. After receiving the data, the processing procedure follows that encompasses the processes of extracting specific information, cleaning, and de-duplicating. The processed data is transmitted to the customized processing module. This customized processing module enables users to process the data according to their own specific needs, such as normalizing integral elements to some value or reducing the dimensions of a record to decrease the scale of the data. It principally requires users to implement the code by programming manually [12].

Furthermore, with the development of information technology and wireless technique, a vast amount of data is collected via sensors used in electronic devices. This collected data can prove to be of immense use at the real-time if processed appropriately. However, data analysis requires hardcore coding to implement algorithms. Agile modeling and aspect-oriented approach can prove to be an excellent option to implement these types of module. Agile modeling helps to incorporate new idea /option and changing requirements and design aspects whereas AO approach enables to develop extensible programming code for such kind of environment [116].

Aspect-Oriented Programming can also pose to be a good option to implement custom processing module in data storage management of IoT and benefited. Due to this kind of programming technique, addition or deletion of customized functions to the module can be done dynamically without rebuilding the original database code and restarting the running program [12]. AOP is also suitable for current large scale dual reality applications like smart cities and smart factories. Such smart environments produce extensive data in real-time that needs lots of development as well as maintenance efforts. Maintenance of such smart environments requires a plug-in mechanism that can implement cross-cutting concern regarding changing data and services; that supports reusability and extensibility [13].

The major identified literature gaps, along with the proposed solutions, are briefly noted in the next section.

1.4 CHALLENGES ADDRESSED AND SOLUTIONS PROPOSED

A critical glance at the existing literature indicates the following issues need to be addressed towards the building of a software quality model for aspect-oriented systems.

- **Comprehensive and comparative analysis of existing quality models and metrics:** *Aspect-oriented approach is relatively a new paradigm built based on Object-Oriented approach. It aims at reduction of the scattered and tangled code and hence improves the quality. There is scope for comparative analysis based on quality evaluation between Object-oriented and Aspect-oriented methodologies.*

  Solution: To ensure a comprehensive analysis of the existing quality models and metrics, detailed and exhaustive literature has been reviewed concerning object-oriented and aspect-oriented software system, and their relative comparison has been made. Work is published in [7][17][18].

- **Realistic quality model for Aspect-oriented systems:** *Quality model provides the basis for specifying quality requirements and evaluating quality. Existing Quality models are limited to software quality characteristics for structured, i.e., Module oriented and object-oriented methodologies. There is a need to explore and design quality models for aspect-oriented systems.*

  Solution: The characteristics of Aspect-oriented Methodologies related to the quality of the software has been identified and analyzed. In this approach, a novel quality model has been developed on the guidelines of ISO25010 (latest ISO standard) for Aspect-oriented software systems Work published in [8].

- **Validation of the proposed quality characteristics:** *There is a need for validation of the proposed quality characteristics for defining an objective measure of project quality.*

Solution: To ascertain the accuracy and to enhance the confidence in the proposed quality characteristics, these have been investigated and validated via multivariate decision-making process. In this approach, the Analytical Hierarchical Process (AHP) technique is used as it appropriately fits for the proposed hierarchical model for the software quality. With the use of the AHP technique [19], corresponding weights of the sub-characteristics are evaluated, and their validity is ensured. Work published in [3][108][21] and communicated in [20].

- **Development/Extension of metrics for Aspect-Oriented Systems and validation:** *For the current metrics suite for Aspect-Oriented, the validation of these metrics in real-world software development settings is limited and need to be investigated further for a different environment. Enormous research effort has gone into defining parameters for Object-Oriented methodology but Aspect-Oriented methodology, being a relatively recent paradigm, is having comparatively lesser number of measurement frameworks for evaluating quality characteristics.*

Solution: A novel set of metrics has been developed for aspect-oriented systems. Also, the proposed metric is validated. Published in [21][22] and Communicated in [20]

- **Quantitative measurement of Quality:** *There is a need for measuring various quality attributes for defining an objective measure of software product quality in order to assist the decision making. An objective and quantitative estimate of quality attributes is required as this help to measure the quality of the software product.*

Solution: With the aim to evaluate the proposed quality characteristics, a framework for evaluation is defined. Using the defined framework, the sub-characteristics of quality are evaluated. Also, a correlation is established between the sub-characteristics and the proposed quality characteristic. Published in [21] and communicated in [20].

1.5 ORGANIZATION OF THESIS

This work is divided into seven chapters. The following is an outline of the contents of the thesis:

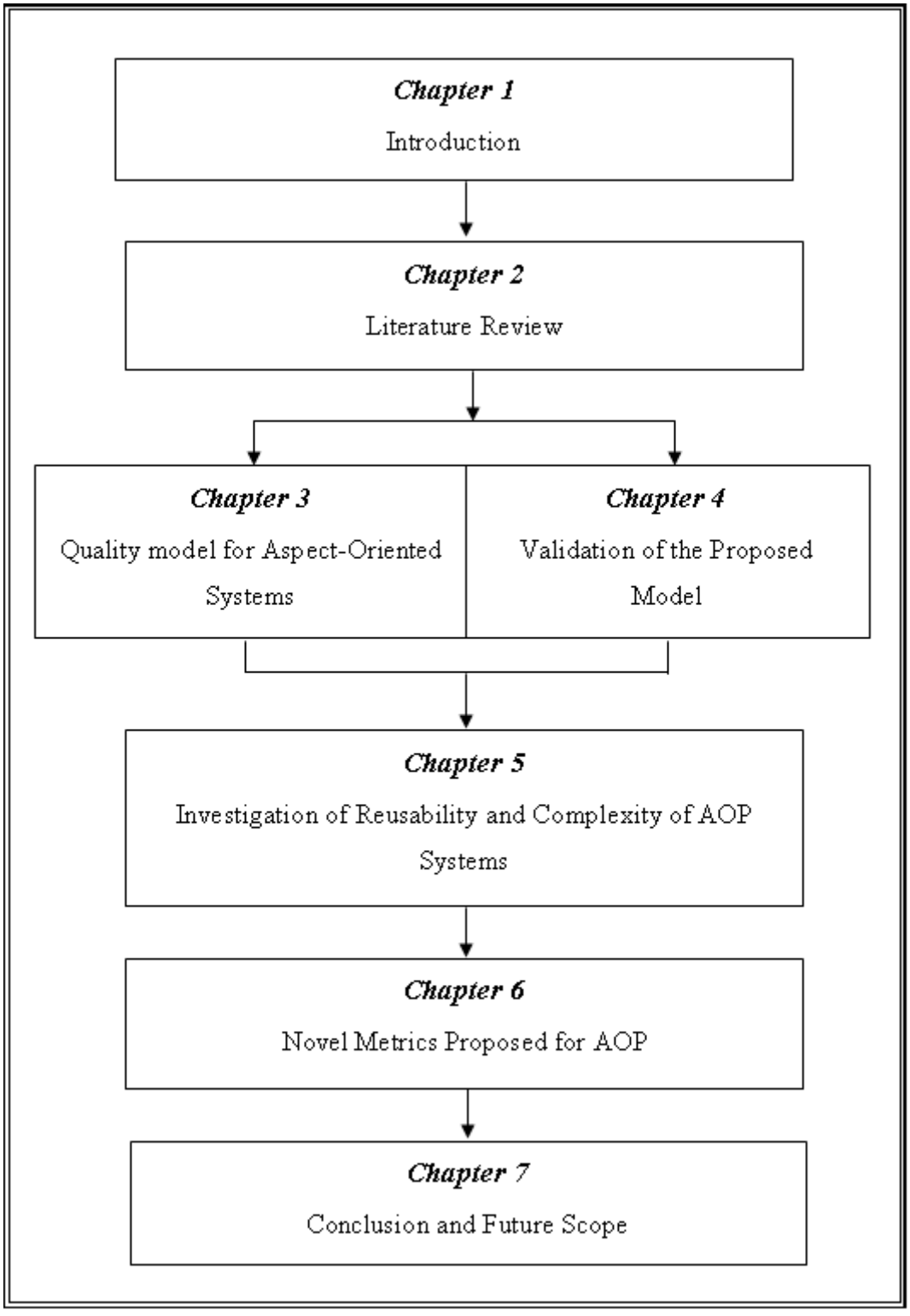


Figure 1.3: Organization of Thesis

❖ **Chapter I**: This chapter commences with the significance of the software quality in today's technological world. Different types of programming methodologies are listed with a particular emphasis on Aspect-oriented programming methodology. The elementary concepts and the architecture of the AOSD are elaborated along with its applicability in modern systems. In the end, the challenges involved and proposed solutions are discussed in this chapter.

❖ **Chapter II**: A comprehensive and exhaustive literature survey of selected publications related to different existing software quality models is carried out. A detailed description of various software metrics for the software quality attributes in different programming methodologies has also been presented. The influence of the AOP metrics on the trustworthiness hence on software quality is studied, and the findings are reported in this Chapter. Also, the various techniques for solving Multi-criteria decision problems have been identified and enumerated. Finally, the chapter wraps up with a summary of the research gaps identified in the literature review

❖ **Chapter III**: In this chapter, a novel software quality model for Aspect-oriented programming methodology is proposed. In addition, the relevance of the proposed sub-characteristics added in the newly proposed model is discussed in detailed.

❖ **Chapter IV**: The validation of the newly added attributes under proposed quality model using Analytical Hierarchy Process (AHP) technique of Multi-Criteria Decision Method (MCDM) is done, and the details are provided in this Chapter.

❖ **Chapter V**: The investigative comparison of the object-oriented program and the aspect-oriented program made using a collection of software metrics is highlighted in this chapter. The details of the statistical comparison of the metrics collected and visualized are presented in this Chapter, along with the result analysis.

❖ **Chapter VI**: This chapter presents three newly proposed metrics for the various quality characteristics concerning aspect-oriented programming methodology. This chapter also presents the empirical and theoretical validation of

the proposed metric by using the exemplary case studies or a range of AspectJ packages accessible as an open-source in the repository of AspectJ or embedded with the Eclipse platform.

❖ **Chapter VII**: Finally, this Chapter summarizes and concludes our contributions and provides guidelines or directions for the future work in this area.

❖ The *bibliography* includes references to relevant publications in this research work.

A brief survey on existing software quality models and metrics is carried out, and their limitations are reported as identified in the following Chapter II. Besides, the impact of AOP on software quality all along with the existing Multi-Criteria decision making methods are discussed in Chapter II.

CHAPTER II

SOFTWARE QUALITY MODELS, METRICS AND MULTI-CRITERIA DECISION MAKING

Literature has been reviewed from four perspectives as presented in Figure 2.1; specifically, existing software quality models, the various quality metrics for the different software programming approaches, particular emphasis on the analysis of aspect-oriented metrics on software quality and the various available multi-criteria decision making methodologies.

2.1 QUALITY MODELS

A software quality model is used to obtain data that help in both specification and evaluation of software quality. Typically, software quality models form a standardized approach to measure software product. That is, software quality models act as a base for evaluation and are used as a means to assess the quality of the software product. It can be used to reassure that the final software product conforms to the expected standards. Different researchers have proposed different software quality models to help measure the quality of software products. There have been various notable models of software quality:-

- Mc Call Quality Model
- Boehm Quality Model
- Standard ISO 9126
- FURPS Quality Model
- Ghezzi Quality Model
- Dromey Quality Model
- Standard ISO 25010

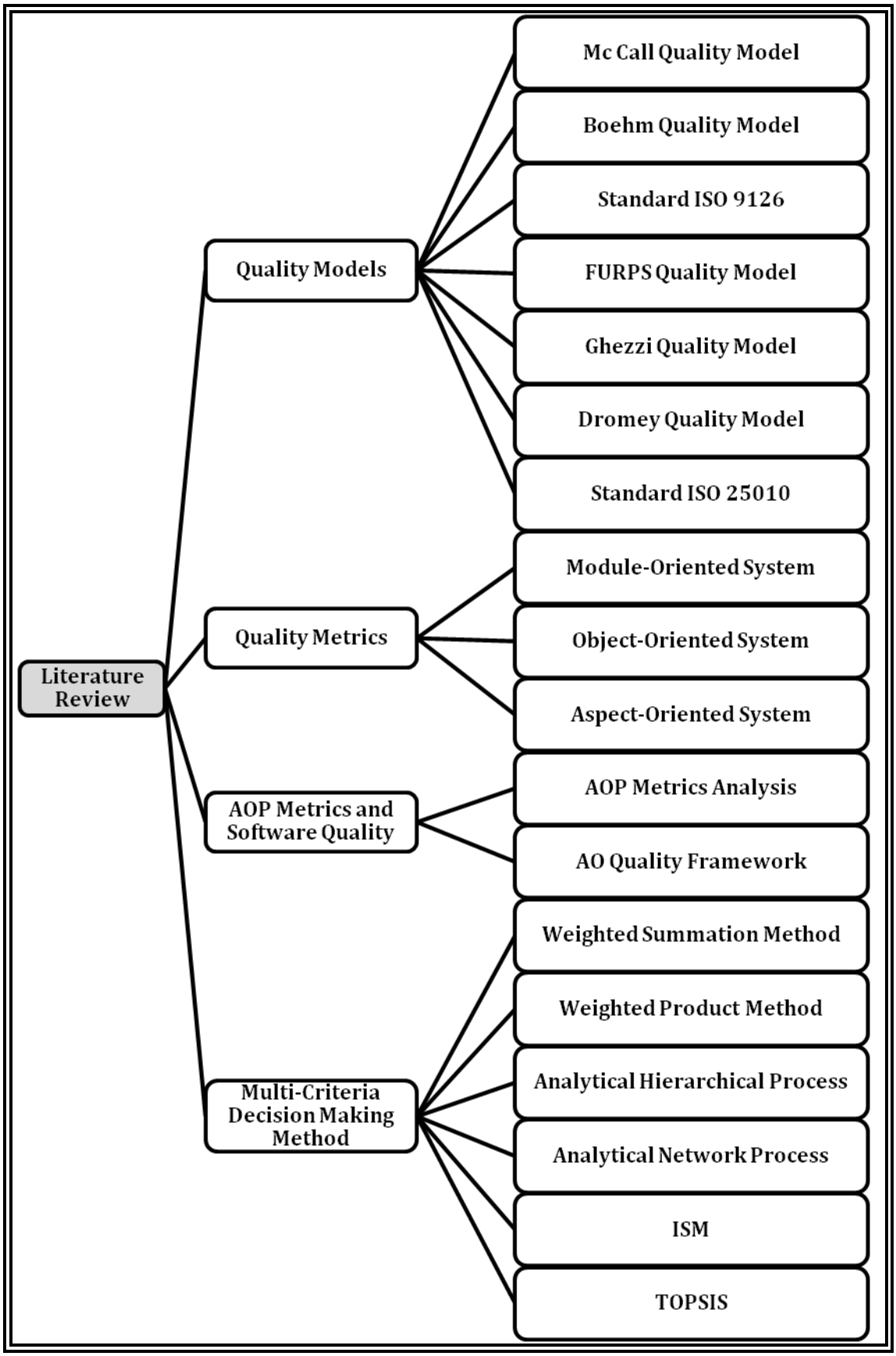


Figure 2.1: Classification of Literature Review

2.1.1 McCall Quality Model

Jim McCall [23] identified three main perspectives for characterizing the quality attributes of a software product, namely product revision, product transition, and product operation. The Mc Call's software quality model identified eleven quality factors (or attributes) that could affect the quality of the software product. These quality factors (or attributes) are categorized under three different categories by keeping three main perspectives in mind.

The first perspective, product operation, is related to functional operations so that the software corresponds to its requirement specifications given by the user.

The second perspective, product transition, is related to the adaptability so that the software is easy to relocate to the new environments.

The third perspective, product revision, is related to change so that the software product is able to make changes itself as per the requirement.

A set of quality criteria or the way of measurements are defined for each of the quality factors of the three perspectives, as shown in Figure 2.2.

A quantitative measure for each quality factor (or attribute) of the software product could be assessed by assessing its corresponding quality criteria. However, to measure overall quality, all specific measures need to be combined by weighted summation of each attribute [24].

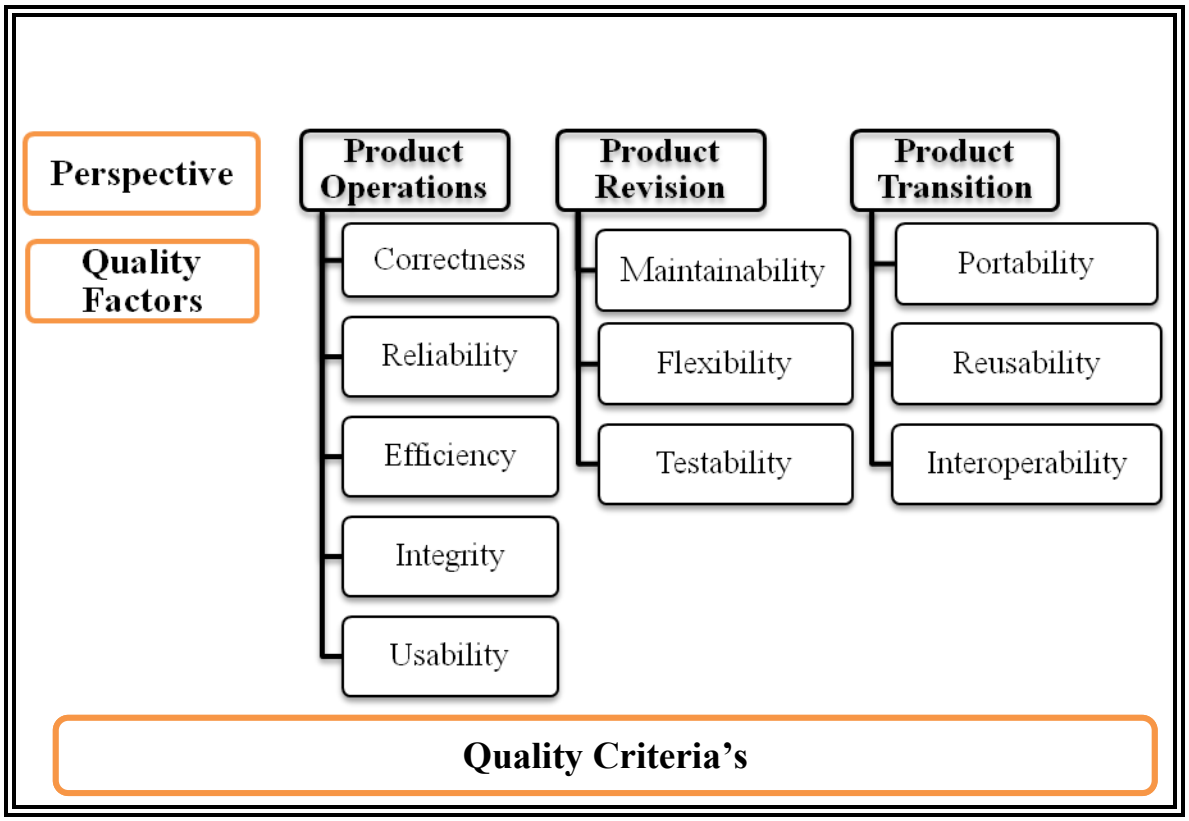


Figure 2.2: Mc Call Software Quality Model

Limitations

The limitations in pretext to Mc Call software quality model are listed as under:

- It is challenging to use Mc Call model to set peruse and specific quality requirements as many of the metrics can be measured only subjectively [2].
- Some of the quality factors cannot be defined or even meaningful at an early stage for non-technical stakeholders.
- This model is not according to the criteria defined in the IEEE standard for software quality metrics methodology for a top to bottom approach to quality engineering.
- Additionally, this model is only specific to product perspective of quality.

2.1.2 Boehm Quality Model

Barry W. Boehm [25][26] defined a gradable model of software quality characteristics, in attempting to qualitatively characterize software quality as a set of attributes and metrics. He identified seven quality factors (or attributes) that could affect the quality of the software product. These quality factors were categorized according to the three primary uses of the software.

First primary use or "general utility" is related to the ease, reliable, and efficient use of software system, termed As–is.

Second primary use or "general utility" is related to the ease of maintenance of software so that it is easy to understand, modify, and retest the software.

Third primary use or "general utility" is related to the change in the environment of the software.

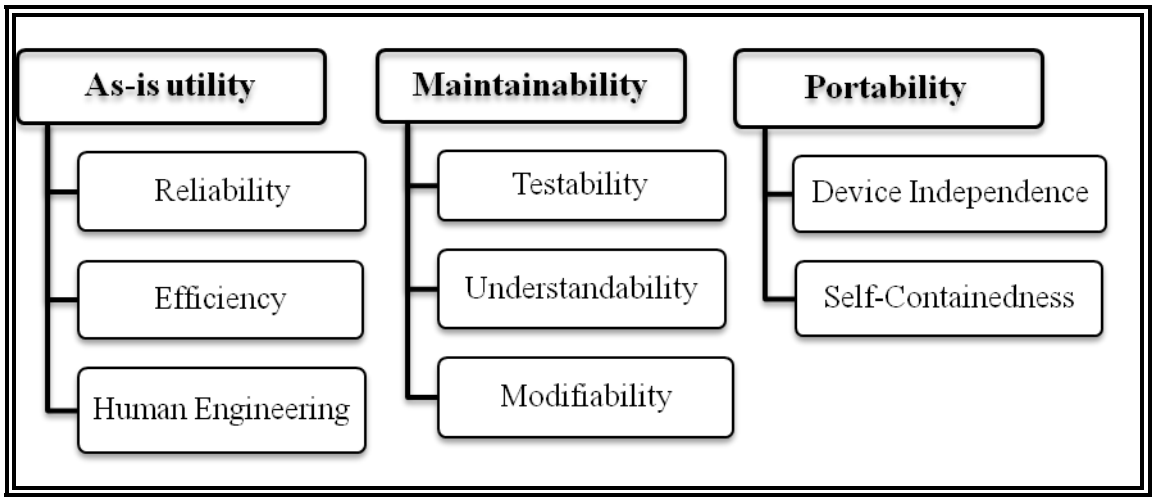


Figure 2.3: Boehm Software Quality Model

Quality factors associated with the primary uses form the next level of Boehm's hierarchical model [24] as listed in Figure 2.3.

Ways of measurements smartphone for each of the seven quality factors of the three essential uses/general utility [24]. These quality factors are further broken down into measurable properties known as primitive characteristics or constructs that may be measured so as to evaluate the overall quality of the software.

Limitations

List of limitations with regards to Boehm software quality model is given as under:

- The primary focus of Boehm's software quality model is only on maintainability.
- Also, in Boehm model, all definitions of the attributes of the software quality begin with "Code possesses the characteristic [….]"; which makes the measurement of quality challenging to understand for non-technical stakeholders at an early stage of software development.

2.1.3 Standard ISO 9126

Initially, the ISO/IEC 9126 [27] is an international standard software quality model that was presented with an idea to standardize the software quality and create a robust framework for assessing the software. Gradually it became an essential criterion for customers to accept the product. Later the extended version of the ISO/IEC 9126 quality model was published consisting of 1 International standard and 3 Technical Reports. It consisted of 2 parts:

- *Product Quality Model*
- *Quality In Use Model*

The ISO quality model [27] presented in the first part of the standard, ISO/IEC 9126-1 categorizes software quality in a structured set of characteristics and sub-characteristics as functionality, reliability, usability, efficiency, maintainability, portability. Each quality sub-characteristic is a result of the presence of some internal software attribute and is externally noticeable when the software is used as a part of a computer system. Hence, each quality sub-characteristic can be further divided into attributes which can be verified or measured in the software product [28].

Product quality model enumerates six quality characteristics related to internal and external quality [27][28] as displayed in Figure 2.4.

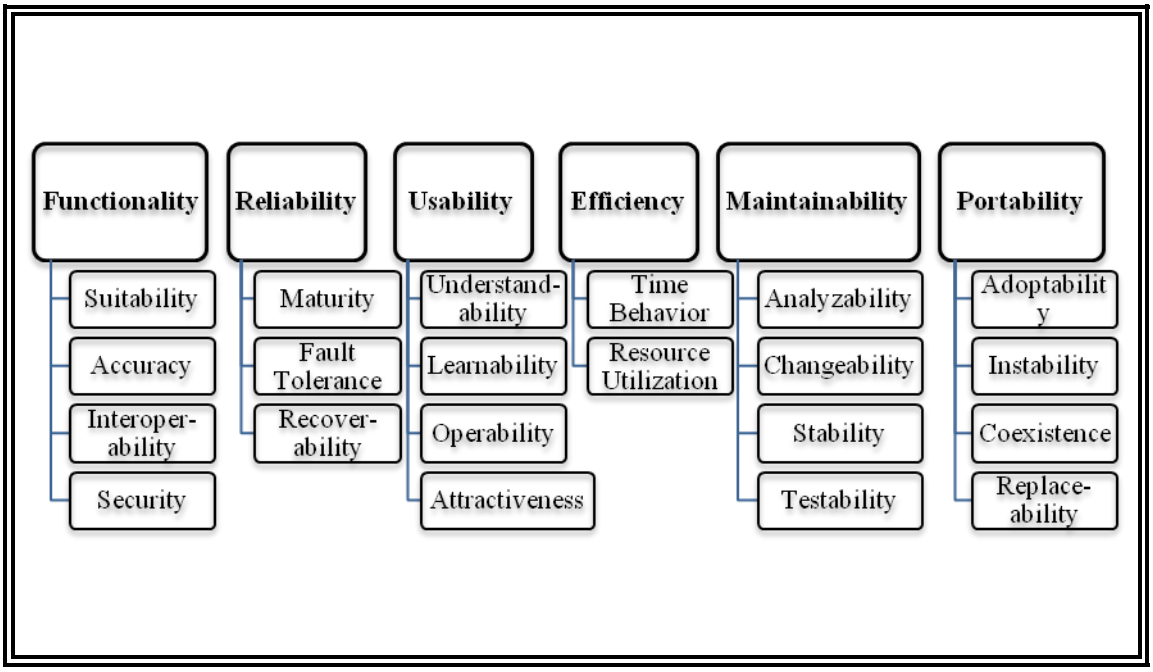


Figure 2.4: ISO 9126 Software Quality Model

Limitations

Various limitations related to ISO/IEC 9126 software quality model are listed as under:

- It describes "what" characteristics should be specified, measured & evaluated, but not "how" these characteristics are to be measured. It specifies characteristics, but no particular metrics were given.
- Security is a vital concern, but it is just a sub-characteristic for functionality characteristic which undervalues its importance.
- There is no sub-characteristic related to the availability of the software system that is an essential feature for software to be reliable.
- Efficiency sub-characteristic is measured only on the basis of resource utilization and time consumed. No emphasis is given on how optimized the code is written, which plays a crucial role in software efficiency.
- For the software to be maintainable and portable, it has to be reversible and extensible, whereas both these features are missing in the current ISO 9126 standard.

- Also, to be maintainable; traceability features also play a significant role which is not present in the ISO 9126.
- Fundamental modularity feature is missing from the functionality characteristics.
- Usability characteristic misses the help and troubleshooting sub-characteristics, which are imperative for a system to be usable.

2.1.4 FURPS Quality Model

Robert Grady [58] at Hewlett Packard presented a hierarchical quality model named FURPS. FURPS is an acronym that represents the five software quality attributes, namely Functionality, Usability, Reliability, Performance, and Supportability. Classical FURPS quality model identified two categories of requirements.

- *Functional Requirements*
- *Non- Functional Requirements*

The Functionality (F) attribute comprises of the functional requirements that are defined by the input and the expected output of the software while the remaining attributes (URPS) comprises of the non- functional requirements that are defined by the design, implementation, interface and physical requirements of the software as displayed in Figure 2.5.

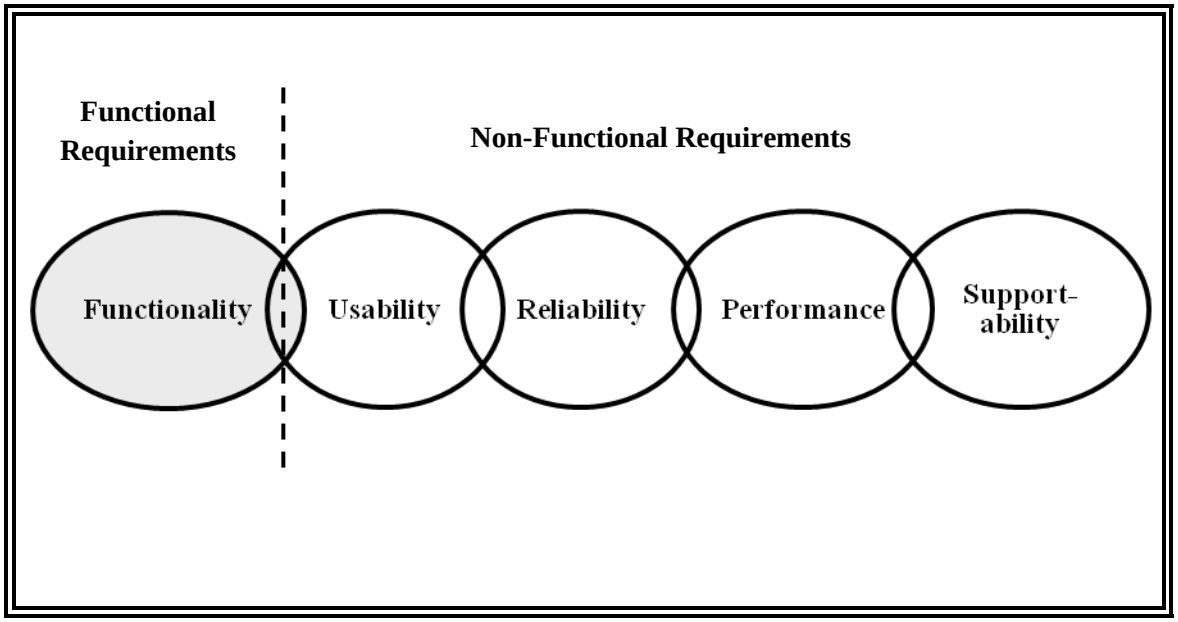


Figure 2.5: FURPS Quality Model

Limitations

Various limitations related to FURPS quality model are enumerated as follows:

- This model is user-centric and disregards the developer's concerns.
- It fails to take into account some of the vital characteristics of the software product like portability, traceability, and maintainability.

2.1.5 Ghezzi Quality Model

Ghezzi Model [60] is built on the idea that the internal qualities dealing with the structure of software can positively assist the software developers in achieving a combined effect both in terms of external as well as internal qualities of software. The overall qualities can be namely accuracy, flexibility, integrity, maintainability, portability, reliability, reusability, and usability, as displayed in Figure 2.6.

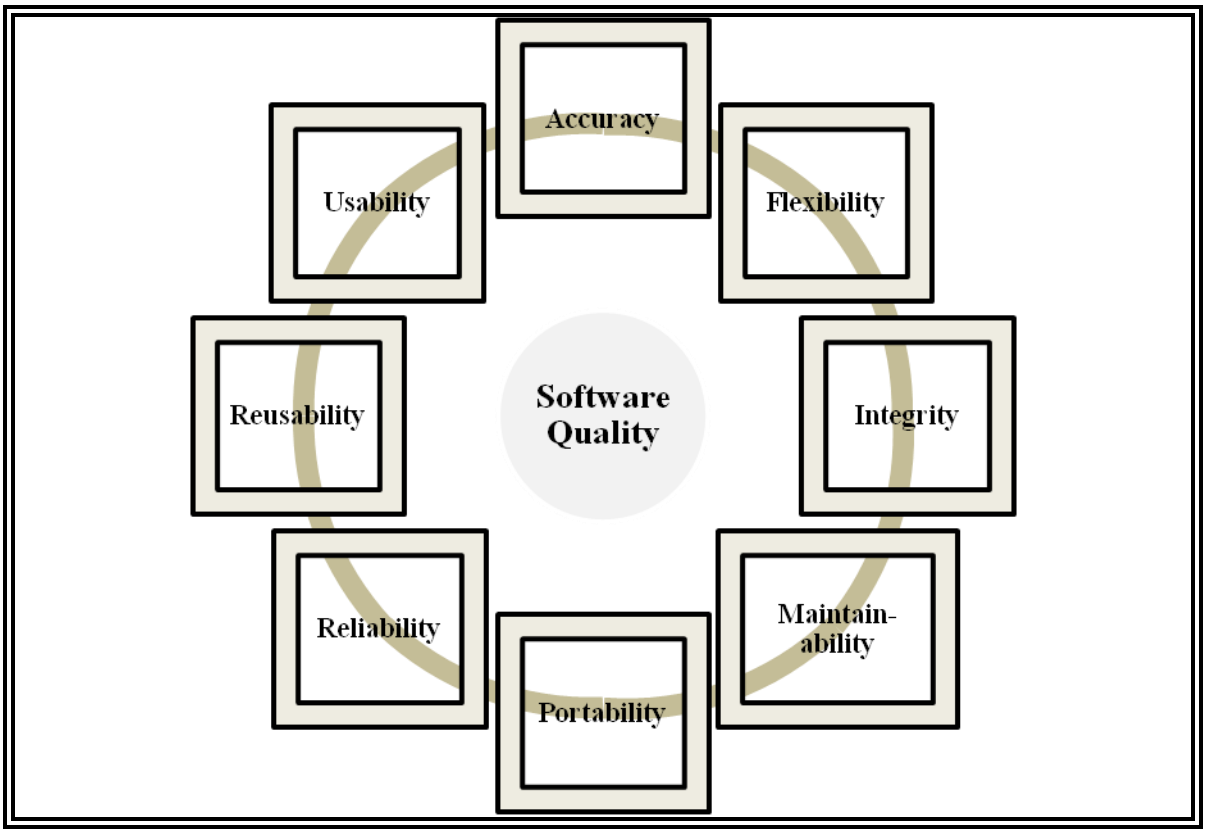


Figure 2.6: Ghezzi Quality Model

Limitations

Various limitations related to Ghezzi quality model are enumerated as follows:

- As Ghezzi quality model deals and focuses mainly on the structure of the software, not the functionality performed by the software hence the primary functionality feature of the software is undervalued in this model.
- This model misses on the testability, robustness, and security attributes of the software.

2.1.6 Dromey Quality Model

Dromey Quality model [68] is designed on the foundation of the relationships that are present in between quality attributes and the software properties. In this model, the emphasis lies in the evaluation of one software's quality with another.

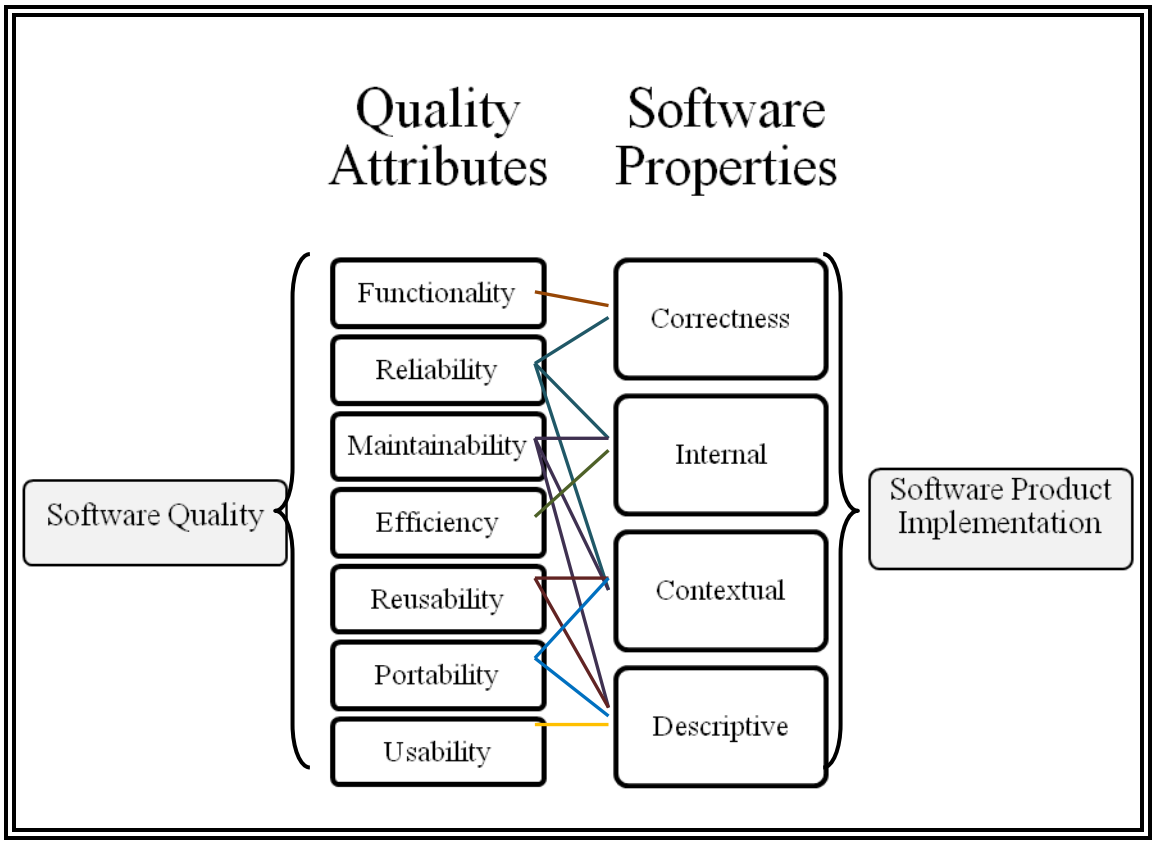


Figure 2.7: Dromey Quality Model

Dromey's quality model is a layered model defined with two layers, as displayed in Figure 2.7.

- *High-level attributes*
- *Software product properties*

The primary advantage of this model is that it helps to find the quality defects in the software product developed and suggest the software properties that need to be reviewed.

Limitations

Various limitations related to Dromey quality model are enumerated as follows:

- Foremost limitation of Dromey's quality model is that it is short of the criteria for measurement of the software quality.
- The model has not been validated for its correctness.
- No standard means is given to measure software product properties.

2.1.7 Standard ISO/IEC 25010

Periodically updating /revision of the software quality models is a significant process as it improves the goodness value and relevance. Reviews include identification of the factors that are found to be substantial as well as the features that have gradually become irrelevant or outdated in the current scenario [29]. The ISO/International Standard Organization and IEC (International Electrotechnical Commission) established a joint technical committee ISO/IEC JTCI, which prepared ISO/IEC 25010 software quality standard [30]. The first edition of this standard, technically revised the ISO/IEC 9126-1:2001 and replaced it.

This International Standard defines a software product quality model to be composed of eight characteristics, which are further subdivided into sub-characteristics that could be measured internally or externally. These are functional suitability, reliability, usability, security, performance efficiency, maintainability, portability, compatibility, as shown in Figure 2.8.

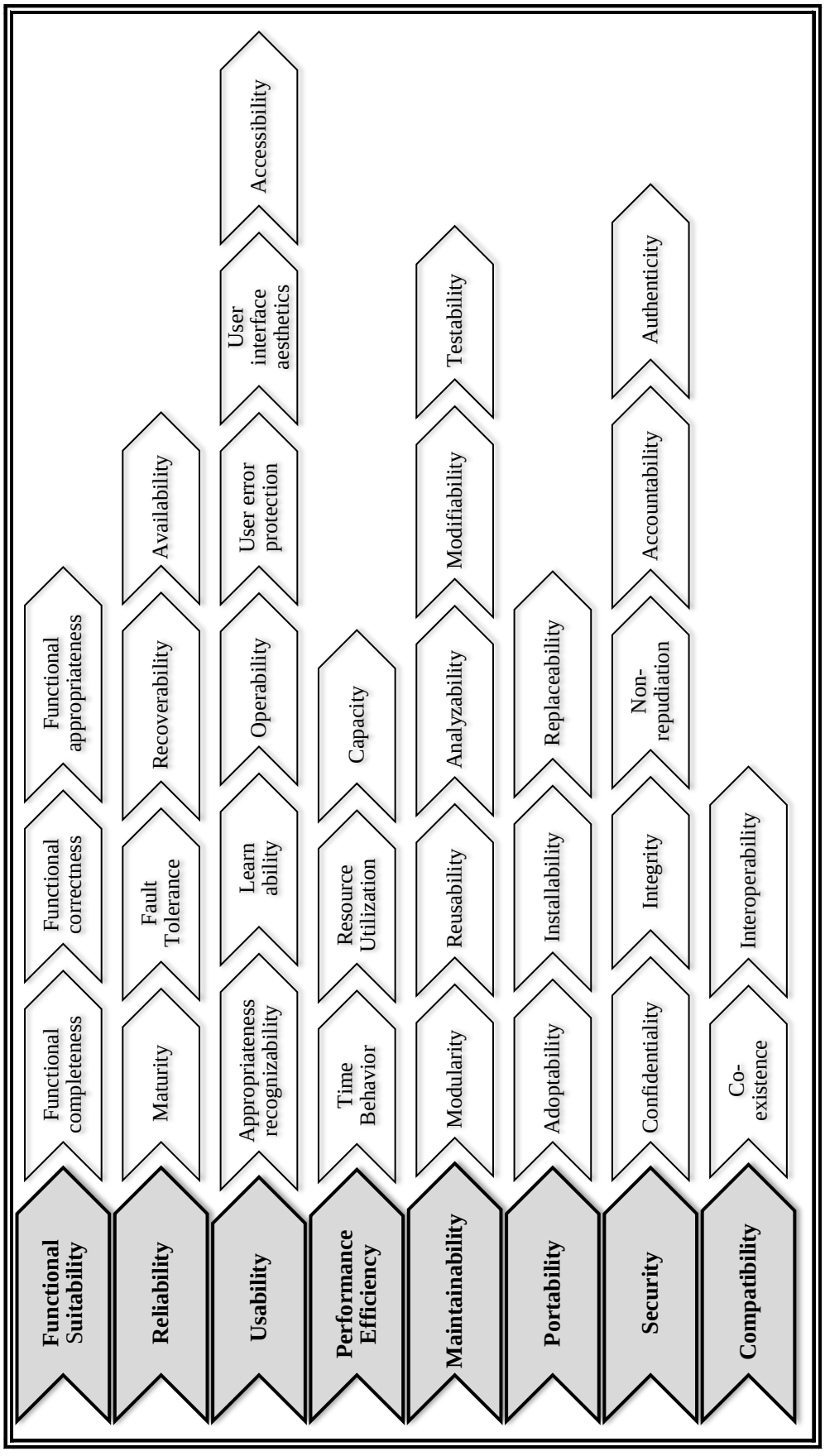


Figure 2.8: ISO 25010 Quality Model

Amendments

Several limitations of ISO/IEC 9126 were tried to be addressed in this new quality model ISO/IEC 25010 with some amendments [30] as enumerated below:

- Revised Product Quality model specifies eight primary quality characteristics instead of earlier six quality characteristics.
- Security, which earlier sub characteristic of functionality is promoted to stand-alone quality characteristic, and is comprised of confidentiality, integrity, non-repudiation, accountability, and authenticity as its sub-characteristics as shown in Table 2.1.

Table 2.1: Security and its sub-characteristics

Characteristics	Sub characteristics
Security	Confidentiality
	Integrity
	Non-repudiation
	Accountability
	Authenticity

- Compatibility feature is added as additional quality characteristic, with Inter-Operability and Co-existence as its sub characteristic as listed in Table 2.2. This characteristic was not there in the previous ISO 9126 software quality standard.

Table 2.2: Compatibility and its sub-characteristics

Characteristics	Sub characteristics
Compatibility	Inter-Operability
	Co-existence

- Two characteristics are renamed in the new ISO25010 software quality model. Efficiency characteristic is renamed to Performance Efficiency and Functionality to Functional Suitability in the new model.
- List of sub-characteristics has been added under Functional Suitability, Performance Efficiency, Usability, Reliability, and Maintainability characteristics as enumerated in Table 2.3.

Table 2.3: List of added sub-characteristics under the corresponding characteristic

Sub characteristic	Characteristics
Functional Completeness	Functional Suitability
Capacity	Performance Efficiency
User error protection	Usability
Accessibility	
Availability	Reliability
Modularity	Maintainability
Reusability	

- Further, the compliance sub-characteristics have been removed from under all characteristics.

Although many drawbacks of ISO 9126 have been addressed in ISO 25010, still there is the scope of improvement as few areas are left unaddressed.

Limitations

The list of constraints pertaining to ISO 25010 quality model is enumerated as under:

- Efficiency characteristic of ISO 9126, although renamed to performance efficiency in ISO 25010, is measured only in terms of resource utilization and time consumed along with the newly added capacity sub-characteristic. Capacity, according to ISO 25010, is the degree to which the maximum limits of a product or system parameter meets its requirements. However, still, no weightage is given to how efficiently the code is written and how much optimized it is.

- Reusability is added as a sub-characteristic to maintainability sub-characteristic, it still misses the extensibility feature which specifies how easily code can be further extended to added functionality/changes.
- Traceability is the main feature of maintainability, which is missing in the new ISO 25010 model too.
- Even, user-error protection and accessibility have been added as new sub-characteristics of usability, but still, it does not includes help and troubleshooting as a sub-characteristic which might turn to be essential for any system to be usable.

Table 2.4 summarizes and compares the various software quality models with regard to the quality attributes that they majorly cover in their respective quality framework.

Table 2.4: Comparison of the various software quality models concerning quality attributes

S. No.	Quality Attribute	Mc Call	Boehm	Standard ISO 9126	FURPS	Ghezzi	Dromey	Standard ISO 25010
1	**Correctness**	✓					✓	✓
2	**Reliability**	✓	✓	✓	✓	✓	✓	✓
3	**Efficiency**	✓	✓	✓			✓	✓
4	**Integrity**	✓						✓
5	**Usability**	✓		✓	✓	✓	✓	✓
6	**Maintainability**	✓	✓	✓		✓	✓	✓
7	**Flexibility**	✓				✓		
8	**Testability**	✓	✓	✓				✓
9	**Portability**	✓	✓	✓		✓	✓	✓
10	**Reusability**	✓				✓	✓	✓
11	**Inter-Operability**	✓		✓				✓
12	**Human Engineering**		✓					
13	**Understandability**		✓	✓				
14	**Modifiability**		✓					✓

S. No.	Quality Attribute	Mc Call	Boehm	Standard ISO 9126	FURPS	Ghezzi	Dromey	Standard ISO 25010
15	**Device independence**		✓					
16	**Self-Containedness**		✓					
17	**Functionality**			✓	✓		✓	✓
18	**Suitability**			✓				
19	**Accuracy**			✓		✓		
20	**Security**			✓				✓
21	**Maturity**			✓				✓
22	**Fault Tolerance**			✓				✓
23	**Recoverability**			✓				✓
24	**Learnability**			✓				✓
25	**Learnability**			✓				✓
26	**Attractiveness**			✓				
27	**Time Behaviour**			✓				✓
28	**Resource Utilization**			✓				✓

S. No.	Quality Attribute	Mc Call	Boehm	Standard ISO 9126	FURPS	Ghezzi	Dromey	Standard ISO 25010
29	**Analyzability**			✓				✓
30	**Changeability**			✓				
31	**Stability**			✓				
32	**Adoptability**			✓				
33	**Instability**			✓				
34	**Coexistence**			✓				✓
35	**Replaceability**			✓				✓
36	**Performance**				✓			✓
37	**Supportability**				✓			
38	**Integrity**					✓		✓
39	**Availability**							✓
40	**Appropriateness Recognisability**							✓
41	**Operability**			✓				✓

2.2 METRICS FOR SOFTWARE SYSTEMS

As a general rule, software quality assurance monitors software engineering processes and methods and hence ensures quality in software with the aid of software quality models. Software engineer requires many Software quality models, such as maintainability, reusability, and reliability, to augment the quality of software [2][31]. Software metrics allow for the measurement of the internal software quality attributes and assists in the analysis, assessment, control, and improvement of software products [32]. Software quality models and software metrics together play a crucial role in software quality measurement. The following section aims to survey various available metrics for the software systems.

Software metrics can be outlined as the continuous application of measurement-based techniques to the software development process and its products to provide meaningful and timely management information, along with the use of those techniques to boost that process and its products.

Software engineer collects the measure and develops the metrics, hence obtain the indicators which lead to informed decision making. By the use of metrics, we can assess the product quality on an ongoing basis that could affectively assist in predicting the cost of maintenance and fault rectification in the same or even similar projects. Hence they could significantly contribute towards the overall improvement of the software as well as the software development process [33].

There are various metrics available to measure the quality attributes depending on the programming methodology used. They are discussed in the following section.

2.2.1 Metrics for Module-Oriented Systems (MOS)

The popular metrics for module-oriented systems [2] are as follows:

a) *Lines Of Code (LOC)*: LOC is defined as the size of the Module Oriented System in terms of Lines Of Code.

b) *McCabe's Cyclomatic Complexity*: This metric measures the complexity of the control flow graph of a method or procedure.

c) Halstead Complexity: This metric computes the complexity on the basis of the number of operators and operands used in the source code.

The list of several module-oriented software metrics along with the respective module-oriented features that they influence are listed in Table 2.5.

Table 2.5: Various MOS features measured in terms of metrics

Feature	Metrics Used
Size	LOC
Complexity	Cyclomatic complexity, Halstead complexity
Maintainability	Number of procedure parameters, Cyclomatic complexity, LOC, length of the user manual
Usability	Number of error messages, Length of the user manual
Reliability	Cyclomatic complexity, LOC, length of the user manual

The next section provides a detailed review of the prevalent metrics for object-oriented software systems.

2.2.2 Metrics for Object-Oriented Systems

The metrics for object-oriented software systems are oriented to the characteristics of object-oriented software like encapsulation, inheritance, polymorphism, information hiding, massaging, localization, and object abstraction techniques. These attributes distinguish the object-oriented software from Module oriented software [34].

A) Chidamber and Kamerer (CK) metrics suite

Chidamber and Kamerer focused on the development of metrics that could be applied onto the classes as these are class-based metrics [35]. In object-oriented System classes are the fundamental means to encapsulate data (attributes) and methods

(operations) into one single unit. The list of several Chidamber and Kamerer object-oriented software metrics along with the particular object-oriented features that they influence are enumerated in Table 2.6.

a) Weighted Methods per Class (WMC): The WMC counts the number of methods implemented in a class or the sum of the complexities of the method (measured by Cyclomatic complexity) and is measured using Equation 2.1.

$$WMC = \sum_{i=1}^{n} c_i \qquad (2.1)$$

$$where\ n = number\ of\ methods$$

b) Response For a Class (RFC): RFC counts the number of methods within a set which can be executed in response to a message sent to an object. It determines the degree of communication between the objects. As the value of RFC increases, the testing, debugging, and the overall maintenance turn into complicated.

c) Lack of Cohesion of Methods (LCOM): LCOM measures the degree of similarity between methods. A higher value of LCOM signifies that the class should break down into two or more sub-classes.

d) Coupling between object classes (CBO): CBO counts the number of other classes to which a class is coupled through its member functions. A higher value of CBO implies excessive dependency on different classes that decrements the modular design and obstructs its reuse. Hence, lower CBO value is desirable.

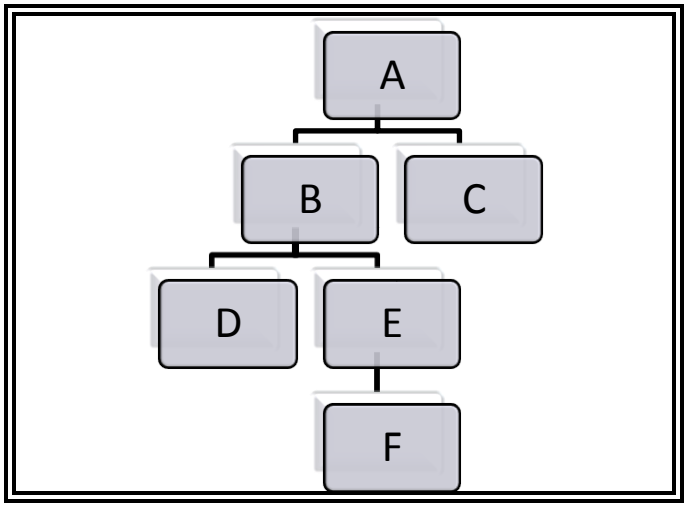


Figure 2.9: Class hierarchy example

e) Depth of Inheritance Tree (DIT): DIT metric measures the length from the class node to the root node (i.e., base class).The deeper the class, more influence on the behaviour from its superclasses, hence increase in complexity. In Figure 2.9, the DIT value is 4.

f) Number of Children (NOC): NOC metric counts the number of subclasses immediately subordinate to a class. As the value of NOC increases, the reusability increases. But alongside the effort required for testing also increases. Hence the ideal value of NOC is relative. In Figure 2.9, class 'B' has NOC value as 2 as it has two children, D, and E that are immediate subclasses of B.

Table 2.6: Various OOS features measured in terms of CK metrics

Feature	Metric used
Encapsulation	**CS**
Message Passing	**COF**
Inheritance	**MIF, AIF**
Polymorphism	**PF**

The objective of the WMC metric is to symbolize the complexity of the complete software system, whereas the rest of the five metrics objective is to indicate the complexity of a specific class. The complexity of a class is an indicator of the amount of effort required for the implementation of testing. Hence WMC should be kept as low as is reasonable.

B) Lorenz and Kidd(LK) metrics suite

Lorenz and Kidd [36] introduced 11 metrics and classified the object-oriented metrics into three broad categories.

- *Class Size*
- *Class Inheritance*
- *Class Internals*

a) *Class Size (CS) Metrics:* CS metrics are a set of six metrics based on a count of the number of operations and the number of attributes in a particular class.

- *Number of Public Methods (NPM)*
- *Number of Methods(NM)*
- *Number of Public Variables per class(NPV)*
- *Number of Variables per class(NV)*
- *Number of Class Variables(NCV)*
- *Number of Class Methods(NCM)*

b) *Class Inheritance Metrics:* Class Inheritance metrics are composed of a set of three metrics based on the number of operations that are inherited from the super-class; the number of inherited operations that are overridden, i.e., redefined by a subclass and the number of newly added operations added by a subclass.

- *Number of Operations Inherited (NOI)*
- *Number of Operations Overridden (NOO)*
- *Number of Added Operations (NAO)*

c) *Class Internal Metrics:* It enumerates the set of two metrics that looks at the general characteristics of the classes.

- *Average Parameter per Method (APM):* APM metric measures the average degree of parameter usage in the method. It is estimated as given in Equation 2.2.

$$APM = \frac{NV}{NM} \tag{2.2}$$

- *Specialization Index (SI):* SI metric measures the degree of specialization for a subclass. It is measured as expressed in Equation 2.3.

$$SI = \left(\frac{NOO * DIT}{NM}\right) \tag{2.3}$$

The list of several Lorenz and Kidd object-oriented software metrics along with the particular object-oriented features that they influence are enumerated in Table 2.7.

Table 2.7: Various OOS features measured in terms of LK metrics

Feature	Metric used
Class	Class Size Metrics
Polymorphism	NOO, NAO
Inheritance	Class Inheritance Metrics and SI

C) Metrics for Object-Oriented Design (MOOD) suite

Abrew et al. [115] presented the list of several object-oriented software metrics as MOOD suite. Later these metrics were embedded in a quality model and empirically validated too. It considered the invisibility of methods as a base for proposing two metrics MHF and AHF. Invisibility of a method is the percentage of total classes from which the specified method is not visible. In this, the inherited methods are not considered.

The metrics composing the MOOD suite are as follows:

a) Method Hiding Factor (MHF): This metric is expressed as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods described in the system under consideration.

b) Attribute Hiding Factor (AHF): This metric is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of methods defined in the system under consideration.

c) Coupling Factor (COF): COF metric is the ratio of the actual number of couplings not imputable to inheritance to the maximum possible number of couplings in the system.

d) Method Inheritance Factor (MIF): MIF metric is calculated as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total

number of available methods for all the classes. Here in this, all methods, which are local as well as inherited are considered.

e) *Attribute Inheritance Factor (AIF):* AIF metric is calculated as the ratio of the sum of the inherited attributes in all classes of the system under consideration to the total number of available attributes (that are local as well as inherited) for all the classes.

f) *Polymorphism Factor (PF):* PF metric is defined as the ratio of the actual number of a possible different polymorphic situation for class Ci to the maximum number of a possible distinct polymorphic situation for class Ci.

MOOD object-oriented software metrics along with the respective object-oriented features that they influence is enumerated in Table 2.8.

Table 2.8: Various OOS features measured in terms of MOOD metrics

Feature	Metric used
Encapsulation	**MHF, AHF**
Message Passing	**COF**
Inheritance	**MIF, AIF**
Polymorphism	**PF**

As these metrics are expressed in a ratio where 0% means ‘no use’ and 100% means ‘max use.’ MOOD metrics suite through experimental data analysis [34][115] is found to be reasonably independent of size.

The discussion of the prevalent metrics for aspect-oriented software systems is covered in the subsequent sections.

2.2.3 Metrics for Aspect-Oriented Systems

With the continuous increase in the dependency on the software, the need for quality software has increased manifold. Growing awareness among the customers has enforced the quality to be measured quantitatively. Metric measurement can be effectively used to assess the quality of the software product and assist in timely decision making. Software metrics are the measurement techniques that are applied to software processes and products for measuring the quality of the software quantitatively to get well-timed and meaningful engineering and management information so as to improvise them. They are considered as the primary indicator of the imperfection detection/prediction in the software process or product and further software maintenance. It is concerned with calculating a numeric value for an attribute for software process/product.

Table 2.9: AOP metrics proposed by various authors for aspect-oriented features

Author	#Metrics	Coupling	Cohesion	Comple-xity	Cross-cutting	Inherit-ance	Size
Zhao and Xu	10 +6 metrics	✓	✓				
Ceccato and Tonella	10 metrics	✓	✓	✓	✓	✓	
Roberto and Sven Apel	10 metrics				✓		
Sant' Anna et al.	10 + 11 metrics	✓	✓	✓	✓		✓
Kumar et al.	6 +1 +1 metrics	✓	✓	✓			

Several studies have been performed to define and assess metrics for Aspect-oriented software systems. The studies focused on determining the effect of aspect introduction on various primary software quality attributes such as size, complexity, cohesion, and coupling. However, most of the defined metrics focus on specific fields, methods, or advice [117]. An outline of software measures proposed by various authors with respect to aspect-oriented features is given in Table 2.9.

A brief summarization of the existing metrics that have been proposed for the aspects are as follows:

A) Zhao and Xu Metrics

Zhao and Xu [31], based on an aspect dependency graph, proposed cohesion measures for aspect-oriented systems. The measure uses inter-module and module-attribute dependencies. Zhao and Xu defined cohesion measures for aspect-oriented systems. It is based on the aspect dependency graph. They presented two ways for measuring aspect cohesion based on inter-attributes (γ_a), inter-module (γ_m) and module-attribute (γ_{ma}) dependencies. In first way, aspect cohesion for an aspect A is to be defined and represented as 3 tuples as Equation 2.4.

$$\tau(A) = (\gamma_a, \gamma_m, \gamma_{ma}) \quad (2.4)$$

Second way is of measuring aspect cohesion as a whole and is expressed as Equation 2.5:

$$\tau(A) = \begin{cases} 0 & when\ n = 0 \\ \beta * \gamma_m & when\ k = 0\ and\ n \neq 0 \\ \beta_1 * \gamma_a + \beta_2 * \gamma_m + \beta_3 * \gamma_{ma} & others \end{cases} \quad (2.5)$$

Where k = number of attributes in aspect 'A' and n= number of modules in aspect 'A'. Parameter weights of $\beta_1, \beta_2\ and\ \beta_3$is arbitrary.

This approach suggests a sophisticated way to measure aspect cohesion that may be problematic to use in real-world development. Moreover, the generation of such dependency graphs is also time-consuming.

Zhao and Xu [38] also use a similar framework to define measurements for aspect coupling. His coupling measure is defined on the number of dependencies between aspect and classes, that is, attribute-class, module-class, module-method and aspect-inheritance dependencies. The metric focused only on the dependencies between class and aspect. Coupling in between aspects and in between classes are not considered during measurement.

B) Ceccato and Tonella AOP Metrics

Ceccato and Tonella [39] revised the well-known Chidamber and Kemerer's (CK) metrics suite [35] for object-oriented and adapted or extended it to make it applicable for Aspect-oriented. In this metric suite, four coupling metrics were proposed along with measures for cohesion, inheritance, and the crosscutting feature of aspect-oriented programming. They termed classes and aspects as module and methods, advices and introductions as operation. They defined ten different AOP metrics, namely -:

a) *Coupling on Advice Execution (CAE):* CAE is calculated by counting the number of aspects containing advices which are possibly triggered by the execution of operations in a given module.—New metric

b) *Coupling on Intercepted Module (CIM):* CIM is calculated by counting the number of modules or interfaces which are explicitly named in the pointcuts belonging to a given aspect.—New metric.

c) *Coupling on Method Calls (CMC):* CMC is calculated by counting a number of modules or interfaces which have declaring methods that are possibly called by a given module. —Derived from CBO (Coupling Between Objects) OO metric.

d) *Coupling on Field Access (CFA):* CFA is calculated by counting a number of modules or interfaces which have declaring fields that are accessed by a given module.—Derived from CBO (Coupling Between Objects) OO metric.

e) *Response For a Module (RFM):* RFM is calculated by counting to methods and advices which may be potentially executed in response to a message that is received by a given module. -- Derived from RFM (Response For a Method) OO metric.

f) *Lack of Cohesion in Operations (LCO):* LCO is calculated by counting the number of pairs of operations working on different class fields minus the number of pairs of operations working on common fields.—Derived from LCOM(Lack of Cohesion in Methods) OO metric

g) *Crosscutting degree of an Aspect (CDA):* CDA is calculated by counting the number of modules that are affected by the pointcuts and introductions in a given aspect.—New metric

h) *Weighted Operations in Module (WOM):* WOM is calculated by counting the number of operations in a given module.—Derived from WOM OO metric.

i) *Depth of Inheritance Tree (DIT):* DIT is calculated by measuring the length of the longest path from a given module to the class/aspect hierarchy root.—Derived from DIT OO metric.

j) *Number of Children (NOC):* NOC is calculated by counting the number of immediate sub-classes or sub-aspects of a given module.—Derived from DIT OO metric.

However, the limitation of this set of metrics is that they were only applicable to small-sized software (250+LOC).

Bartsch and Harrison [40] evaluated five aspect-oriented coupling metrics by Ceccato and Tonella work, namely CAE, CIM, CFA, CMC, and CDA. It argued that none other than CDA of them are entirely valid without changes from the measurement theory point of view.

C) Roberto and Sven Apel Crosscutting Metrics

Roberto and Apel [41] proposed metrics that could only identify the crosscutting relations and measure and characterize crosscutting in aspect-oriented programming. A range of basic code metrics for measuring and characterizing crosscutting in aspect-oriented programming is proposed. These metrics are categorizing crosscutting into program structure metrics and feature crosscutting metrics.

The program structure metrics highlight the contribution of aspects to the overall structure of programs that are measured in lines of code. Various Program structure metrics are defined as follows:-

a) *Number Of Features (NOF):* NOF is calculated by counting the number of features in a program.

b) *Number Of Aspects (NOA):* NOA is calculated by counting the number of aspects in a program.

c) *Number of Classes and Interfaces (NCI):* NCI is calculated by counting the number of classes and interfaces in a program.

d) *Base Code Fraction (BCF):* BCF is calculated by the number of lines of code that comes from standard Java classes and interfaces relative to the lines of code in a program.

e) *Aspects Code Fraction (ACF):* ACF is calculated by the number of lines of code that come from aspects relative to the lines of code in a program.

f) *Introductions Fraction (IF):* IF is calculated by the number of lines of code that come from introductions or inter-type declarations relative to the lines of code in a program.

g) *Advice Fraction (AF):* AF is calculated by the number of lines of code that come from pieces of advice relative to the lines of code in a program.

However, the feature crosscutting metrics adapt the concept of homogeneous concern and heterogeneous concern to features and provide quantitative criteria to classify within a spectrum that goes from homogeneous to heterogeneous according to the number and type of crosscuts they implement.

a) *Feature Crosscutting Degree (FCD):* FCD is calculated by counting the number of classes that are crosscut by all pieces of advice in a feature and by the Introductions.

b) *Advice Crosscutting Degree (ACD):* ACD is calculated by counting the number of classes that are crosscut exclusively by the pieces of advice in a feature.

c) *Homogeneity Quotient (HQ):* HQ is calculated by the division of the advice crosscutting degree by the feature crosscutting degree.

d) *Program Homogeneity Quotient (PHQ):* PHQ is calculated by the summation of the homogeneity quotients for all the features in a program, divided by the number of features.

These metrics take only crosscutting relations into account which are generated by pointcut shadows and introductions.

D) Sant Anna et al. Metrics

Sant Anna et al. [43] proposed a framework based on a suite of metrics and quality model to assess the reusability and maintainability characteristics for AOSD. The metrics suite was composed of 5 design metrics and 5 code metrics, which were based on the separation of concerns, coupling, cohesion, and size attributes.

Sant Anna et al. [44] also proposed a concern driven measurement framework to assess the modularity of the software architecture. The framework includes the suite of metrics based on concerns along with the way to document the concerns in the architecture. The metric suite was composed of 11 metrics in 4 categories, namely complexity, coupling, cohesion, and separation of concern.

Since the obtained metric value is not normalized to a specific range, hence it is difficult to interpret from the obtained results.

E) Kumar et al. AOP Metrics

Kumar et al. [45] studied the connections that lead to the coupling in aspect-oriented software systems. They included HyperJ, CeaserJ, and AspectJ in their framework. They identified 23 types of connections between the relevant elements for the coupling measure.

Kumar et al. [47] proposed six coupling metrics for the generic aspect-oriented systems namely *Coupling on Attribute Type (CoAT)*, *Coupling on Parameter Type (CoPT)*, *Coupling on Attribute Reference (CoAR)*, *Coupling on Operation Invocation (CoOI)*, *Coupling on Inheritance (CoI)* and *Coupling on High Level Association (CoHA)*.

Also, they proposed one cohesion metric [48], *Unified Aspect Cohesion (UACoh)* based on connections between the members of the component. The proposed cohesion metric is a generic and unified metric that means, the metric is applicable to most of the AOP languages. Six different types of connections are identified that could affect cohesion. The UACoh is defined as in Equation 2.6:

$$UACoh(C_i) = \frac{ANC(C_i)}{MNC(C_i)} \in [0,1] \quad (2.6)$$

Where

ANC = actual number of connections,

MNC = maximum number of connections

The *complexity metric of the aspect-oriented system (CMPX)* [49] is also proposed. CMPX is identified as dependent on two factors present in the component, namely

- *code complexity*
- *interaction complexity*

Code complexity of an aspect-oriented component (class / aspect) is considered to be due to the complexity of the attributes, operations (methods / advice) and nested components present in the component.

Interaction complexity of an aspect-oriented component (class / aspect) is considered to be due to invocation of the operations, reference to the attribute and due to the execution of the statements that cause an interaction between the components that is between classes, between aspects and between class and aspect.

Kumar et al. [46] extended the work to ascertain the correlation between UACoh value and changeability. The findings conclude that UACoh cohesion metric could not be used as an indicator for assessing changeability of aspect-oriented software system.

The proposed framework of metrics has not been empirically validated and has not been applied to case study applications.

Although a lot of research effort has gone into defining metrics for Object-Oriented methodology but Aspect-Oriented methodology, being a relatively new paradigm, is having comparatively lesser number of extensive measurement frameworks [47][50]. The next section describes the brief summary of the prevalent Aspect-Oriented metrics along with their influence on the quality of the software to better understand the concept and the relevance.

2.3 AOP METRICS AND SOFTWARE QUALITY

Trustworthy software systems form the basis for quality software systems. Trustworthiness is the ability of the system to produce expected results despite all odds. It is the assurance that the trusted system will imbibe into the customers, that despite environmental disruptions, intentional or unintentional faults or attacks, it will perform as expected. Various characteristics are covered under trustworthiness like correctness, security, privacy, safety, survivability, and quality of service. Each characteristic acts as a pillar to develop trustworthy software systems. Presence of the characteristics adds to the trustworthiness while its absence decreases the trustworthiness. Each characteristic can have sub-attributes, which can be assessed using corresponding software metrics [14][15].

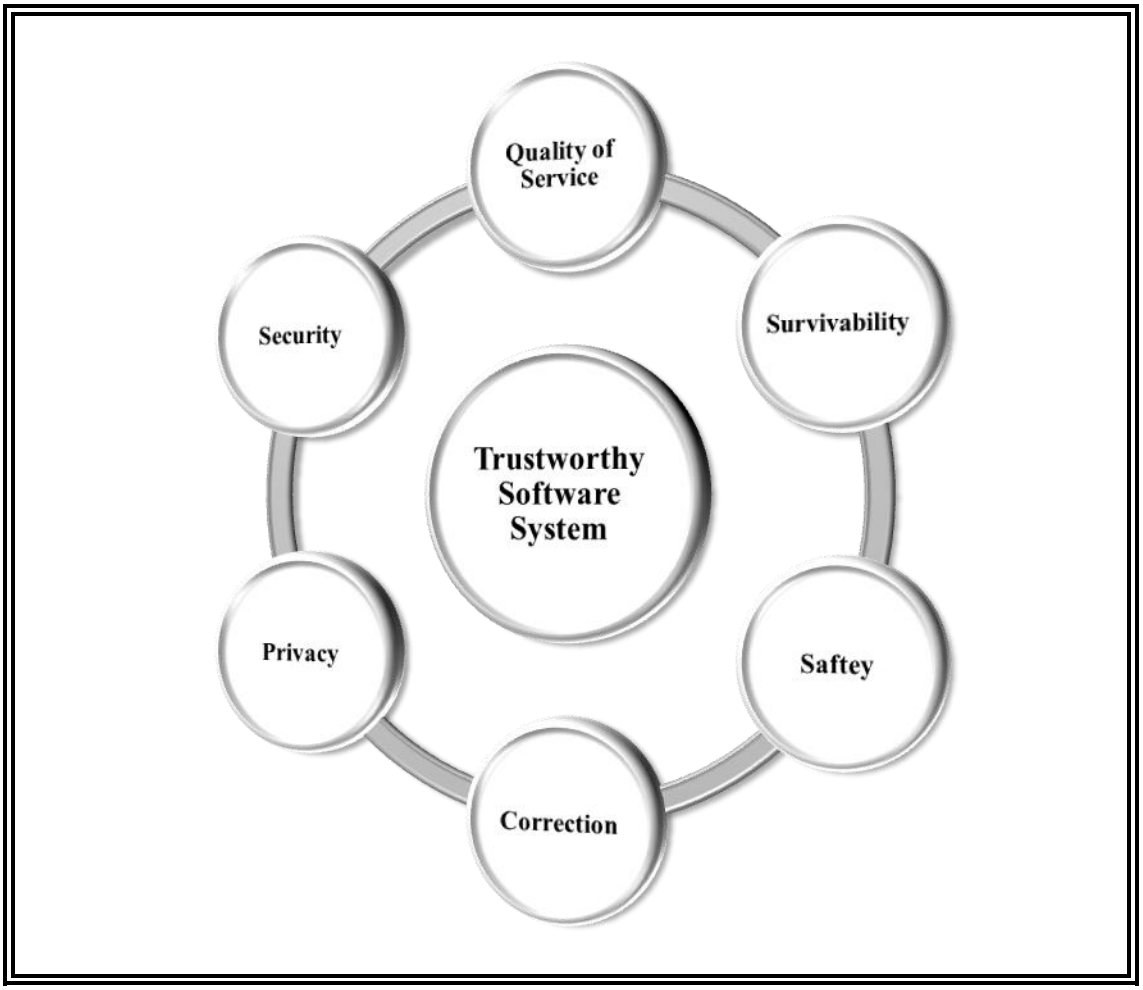


Figure 2.10: Attributes of Trustworthy Software

Developing trustworthy software is the new focus of software developers, along with the functional software. Software metrics are the way to quantify the qualitative

attributes of the software. Various researchers have proposed different software metrics to measure the numerous attributes of the software. The aspect-oriented software development approach is a relatively new approach for software development that is gaining attention; hence, a few software metrics are proposed and validated till date. Separation of concerns is an important concept and activity in structuring the software systems meaningfully and can play an essential role in adding the trustworthiness to the software.

Trustworthiness can be considered as a subset of software quality that adds confidence into the customer towards the software product. Security and dependability are regarded as the fundamental pillars to build trustworthiness. The programming language chosen do affect these two features and ultimately on trustworthiness and quality. The various attributes identified to influence the trustworthiness of the software are shown in Figure 2.10. In aspect-oriented programming, the focus is on to proper handling of scattered and tangled code and increase the modularity using the design unit called Aspect. Safonov [16] in his book demonstrated the typical trustworthy concerns like security checks; multithreaded safety is implemented using Aspect-oriented programming. Accordingly, aspect-oriented programming is an adequate tool to implement trust in the software.

To investigate, which software metrics help assess the quality of aspect-oriented software, a systematic investigation have been conducted and hence the relationship of the AOP metrics with quality is analyzed. Overall 65 AOP metrics based on aspects, join points, pointcuts, introductions, etc. for aspect-oriented programming approach have been collected and their connectivity with the overall software quality is analyzed. Also, in the discussion of the metrics, various metrics are identified to affect precisely complexity, extensibility, reusability, encapsulation, and understandability of the Aspect-oriented Software.

Various Aspect-Oriented Metrics proposed by the researchers have been studied, and its effect on the trustworthiness [17] and hence on quality of the aspect-oriented software are inspected and enumerated in Table 2.10.

Table 2.10: Relevance of the AOP metrics with the quality attributes

S.No.	Author	Title	#Metrics	Metric	Feature
1	**Zhao and Xu [38]**	Measuring Coupling in Aspect-Oriented Systems	10 metrics in 4 categories	1. attribute-class dependency	Coupling
				2. module-class dependency (advice-class, intertype-class, method-class, and pointcut-class dependency)	
				3. module-method dependency (advice-method, intertype-method, method-method, and pointcut-method dependency)	
				4.aspect-inheritance dependency	
2	**Zhao and Xu [37]**	Measuring Aspect Cohesion	6 metrics in 3 categories	1. inter-attribute dependency	Cohesion
				2. inter-module dependency	
				3. module-attribute dependency (advice-attribute, intertype -attribute, method-attribute, and advice-attribute)	
3	**Ceccato and Tonella [39]**	Measuring the Effects of Software Aspectization	10 metrics	WOM	Complexity
				DIT, NOC	Inheritance

				CAE,CIM,CMC,CFA	Coupling
				RFM	Message communication
				LCO	Cohesion
				CDA	Crosscutting nature
4	**Roberto and Sven Apel et al. [41]**	Measuring and Characterizing Crosscutting in Aspect-Based Programs: Basic Metrics and Case Studies	10 metrics in 2 categories	1.program structure metrics (NOF,NOA,NCI,BCF,ACF, IF)	Crosscutting nature
				2. feature crosscutting metrics (FCD,ACD,HQ,PHQ)	
5	**Sant' Anna et al. [43]**	On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework	10 metrics in 4 categories	1. SoC metrics (CDC,CDO,CDLOC)	Concern
				2. Coupling metric (CBC,DIT)	Coupling
				3. Cohesion metric (LCOO)	Cohesion
				4. Size metric (VS,LOC,NOA,WOC)	Size
6	**Sant' Anna et al. [44]**	On the Modularity Assessment of Software Architectures: Do my architectural concerns count?	11 metrics	CDAC,CDAI,CDAO	Concern
				CIBC,IIBC,OOBC,AC,EC	Coupling

				LCC	Cohesion
				No. Of Interfaces, No. Of Operations	Complexity
7	**Kumar et al.[47]**	Generalized Coupling Measure for Aspect-Oriented Systems	6 metrics	Coupling on Attribute Type(CoAT), Coupling on Parameter Type(CoPT), Coupling on Attribute Reference(CoAR), Coupling on Operation Invocation(CoOI), Coupling on Inheritance(CoI) , Coupling on High Level Association (CoHA)	Coupling
8	**Kumar et al.[48]**	Unified Cohesion Measures for Aspect-Oriented Systems	1 metric	Unified aspect cohesion (UACoh)	Cohesion
9	**Kumar et al.[49]**	A Fuzzy Logic Approach to Measure Complexity of Generic Aspect-Oriented Systems	1 metric	Complexity metric of the aspect-oriented system ($CMPX_{AOS}$)	Complexity

2.3.1 AOP Metrics Analysis

Various metrics proposed in the literature have been analyzed to see the impact of aspect orientation on software development metrics and explore the effect on trustworthy characteristics and hence, quality and summarized in Table 2.10. The term *module* applies to both classes & aspects as the metrics are either adapted or extended and are applicable to both the modularisation units. Similarly, the term *operation* applies to both methods of the class and advises/ introductions of the aspect.

1) CAE (Coupling on Advice Execution):

This metric depicts the coupling between the given module and the aspect containing advice which may alter the behavior of the operation. Higher values of CAE indicates high coupling of the aspects with the given module and low accessibility.

2) CIM (Coupling on Intercepted Module):

This metric captures the direct knowledge an aspect has about the rest of the system. A higher value of CIM indicates high coupling of the aspect with the application and hence low reusability.

3) CMC (Coupling on Method Call):

This metric depicts coupling between the module and the methods from different modules. A higher value of CMC indicates the higher dependency from the methods of other modules to the given module. As a functionality of the given module cannot be readily isolated from the other modules; hence, higher values lead to low reusability.

4) CFA (Coupling on Field Access):

This metric depicts coupling between module & the fields from different modules. A higher value of CFA indicates the higher dependency from the fields of other modules to the given modules. As the aspect can access class fields to perform their function; hence, higher values lead to low extensibility.

5) RFM (Response For a Module):

This metric depicts the potential communication between the given module and the other modules. In AOP software, implicit responses may be generated due to pointcut interception, increasing the complexity hence lower understandability.

6) LCO (Lack of Cohesion in Operations):

This metric depicts the cohesiveness of the operations within a given module. Lower values of LCO indicates all the operations in a given module are sharing a standard data structure hence promotes encapsulation.

7) CDA (Crosscutting Degree of an Aspect):

This metric depicts all the modules that may be possibly affected by an aspect. Hence it gives the overall impact of an aspect on the other modules. Higher the value of CDA indicates the degree of generality of an aspect therefore easily extensible.

8) WOM (Weighted Operations in a Module):

This metric depicts the internal complexity of a module in terms of a number of implemented functions. A higher value of WOM indicates more application-specific module, limiting the extensibility.

9) DIT (Depth of Inheritance Tree):

This metric depicts the scope of the properties of the class. More is the value of DIT indicates deeper class/ aspect thus more complex to understand & change.

10) NOC (Number Of Children):

This metric depicts the scope of the properties of the class but in the opposite direction. Higher the value of NOC indicates more modules potentially dependent on the properties inherited from the given module hence increased generality, thereby increased reusability.

11) PHQ (Program Homogeneity Questioned):

This metric is calculated by the submission of HQ (homogeneity questing). For all the features in the program divided by a number of features as PHQ tends to value ‘1’ then it means the program is making full use of the crosscutting capabilities of advice. While PHQ tends to value ‘0’, then it may have one of two interpretations. Firstly, the

majority of class pointcuts are due to ITD's. Secondly, majority of the features have zero pointcuts.

To analyze the current status of the research concerning Aspect-oriented based quality model and metrics; various software quality measurement frameworks proposed on the basis of multiple techniques are being studied in the subsequent section.

2.3.2 Aspect-Oriented Quality Frameworks

The highlighted summary of the aspect-oriented quality frameworks studied are listed as under:

Haijun Yang et al. [84] proposed a software product quality assessment with ISO standards based on the fuzzy logic technique. It defined the establishment of the software quality assessment system using fuzzy measures to quantize fuzzy characteristics and then applied the Choquet integral for synthetic evaluation. Fuzzy measuring is used for the maintainability, reliability, and related costs of lines of code and the function point, which is a single characteristic of software quality. The assessment index and corresponding importance index are adjusted during the implementation phase to make the evaluation more accurate, scientific, and realistic.

Kevin Kam et al. [85] proposed a Fuzzy Group Analytical Hierarchy Process for evaluating the quality of software, with judgments by a group of experts at different levels. The international standard of software quality attributes, which contains 6 criteria with 27 sub-criteria, is applied to the attributes of software quality. The method can help various experts, including developers, testers, and purchasers, to measure the level of the software quality for in-house development or third-party development.

Adesh Kumar Pandey et al. [86] proposed component-based software development using the Analytical Network Process (ANP) to solve decision problems. ANP is a decision analysis technique that reduces the dimensionality of problems. The model is used to calculate the numeric value of the quality of the software component in the biometric domain.

Ural Erdemir et al. [87] proposed a graph-based object-oriented software quality visualization tool called E-Quality. E-Quality automatically extracts quality metrics and class relations from Java source code and visualizes them on a graph-based interactive visual environment. This visual environment effectively simplifies the comprehension and refactoring of complex software systems. This approach helps developers in the understanding of software quality attributes by level categorization and perceptive visualization techniques. It provides a novel visualization for understanding software quality attributes by level categorization and an intuitive visualization technique.

Adam Przybyłek et al. [88] described a quasi-controlled experiment that compared the evolution of two functionally equivalent programs developed in two different paradigms. The study aimed to explore the claim that software developed with aspect-oriented languages is easier to maintain and reuse than software implemented with object-oriented languages.

Ananthi Sheshasaayee et al. [89] proposed a theoretical framework to build maintainability model for aspect-oriented systems. The framework uses a set of static metrics to calculate the quality attributes for aspect-oriented software. Thereafter, based on the collected metrics, an aspect-oriented maintainability model is derived. Further, in [90], a fuzzy logic-based algorithm for software maintainability assessment of aspect-oriented systems is proposed.

Pradeep Kumar et al. [91] proposed a fuzzy logic-based framework is proposed to assess the reusability of aspect-oriented systems. Separation of concern (SoC), cohesion, coupling, size, and complexity are identified as input variables in the fuzzy model. Rules were designed based on the input variables. Reusability is estimated based on firing some rules on the proposed fuzzy-based AO reusability system.

Puneet Jai Kaur et at [92] proposed a package level metric for aspect-oriented system and evaluated reusability of a package. The theoretical and empirical validation of the metric is done in [93]. Also, the impact of the package level cohesion on reusability measure is established using a correlation method. Finally, a framework for assessing reusability using package-level cohesion measure in the aspect-oriented system is proposed.

Blaschek et al. [94] published a patent on presenting a method and device for automatic evaluation of the quality of the software source code. The pre-set of evaluation rules and/or metrics is used for evaluation purposes. At least one embodiment of the pre-set is adapted in accordance with the evaluation of an inspection performed on the source code. Adapted set of evaluation rules and/or metrics different from the first set formed can be used to carry out a modern control of the internal software quality.

Nir-Buchbinder et al. [95] designed a patent for cross-concern code coverage assessment. It presented the method of software quality assessment using Meta information analysis. During the execution of the elements, at the time of test run, meta-information with respect to code elements may be generated, and the coverage of these items may be evaluated. The processor extracts this Meta information and assigns respective metrics for quality indication.

Sarkar et al. [96] issued a patent on measuring the quality of software modularization. Modularization evaluator is presented to be used to determine the quality and/or degree of software source code modularization. Quality of the modularization is evaluated using structural modularity, architectural modularity, size and similarity of purpose perspective. The amount of changes done is incorporated so that the degree of code enhanced or damaged modularization can be tracked.

Burrows et al. [42] reviewed various AOP maintainability studies and analyzed whether most frequently used AO coupling metrics effectively measured the attributes regarding maintainability. This study found that *coupling between components (CBC)* and *depth of inheritance tree (DIT)* are the most common metric which has appeared in nearly 66% of the studies. Some studies have also used coupling metrics especially designed for AOP like *coupling on advice execution (CAE)* and a number of *degree diffusion pointcuts (dPC).* However, the drawback was that these metrics are only based on outgoing coupling connections (fan-out). Various other metrics used are *response for a module (RFM), number of children (NOC),* and *number of indifferent concerns (InC).* It was also found that the majority of AO metrics suite (extended or new) did not focus on interfaced complexity.

From the literature survey done on quality and its characteristics framework for aspect-oriented software systems, it is identified that efforts have been made to design frameworks for overall quality assessment, especially for maintainability, reusability, and modularization. No framework has been designed for assessing extensibility and supportability feature for aspect-oriented systems.

The following section provides a review of the literature on the application of the various identified Multi-criteria decision making approaches.

2.4 MULTI-CRITERIA DECISION MAKING

Quality model, like every model, is composed of several defined characteristics. When multiple objectives are significant to decision making, then a multi-criteria decision making approach comes to rescue. Multi-criteria decision making is a potential tool that is used for analyzing multifaceted problems. It has the ability to evaluate and pick from the different available choices or alternatives, on the basis of various decisive factors or criteria, for the probable selection of the best appropriate choice or alternative. There are three necessary steps involved in utilizing any of the decision making technique that involves quantitative analysis of the alternatives.

Step 1: Determine relevant criteria and alternatives.

Step 2: Calculate quantitative value to determine the relative importance of the criteria and the impact of alternatives on these criteria.

Step 3: Assess the ranking of each alternative.

There are a number of Multi-criteria decision making techniques available in the literature. A fine literature review reveals that specific MCDM methods are better appropriate for certain applications while a few for some other applications. A few of the various techniques available that work as a powerful tool to solve various complex, multi-dimensional decision-making (MCDM) real-world problems are:

- Weighted Summation Method (WSM)
- Weighted Product Method (WPM)

- Analytical Hierarchical Process (AHP)
- Analytical Network Process (ANP)
- Interpretive Structural Modelling (ISM) Approach
- TOPSIS

2.4.1 Weighted Summation Method

The Weighted Sum Method [118] is one of the initial and most frequently used methods for one-dimensional problems. For multi-dimensional problems where a number of criteria and alternatives exist, the WSM is extended on the basis of additive utility assumption. Accordingly, the WSM measure, as given in Equation 2.7 in order to calculate the summated value, is evaluated for each alternative as:

$$WSM_i = \max \sum_{j=1}^{m} w_j * x_{ij} \qquad for\ i = 1\ to\ n \qquad (2.7)$$

Where WSM_i is the Weighted Sum Measure for 'i^{th}' alternative, m is the number of criteria, n is the number of alternatives, w_j is the weight of j^{th} criteria, x_{ij} is the score of i^{th} alternative concerning j^{th} criteria.

The total score that is a weighted sum measure of each alternative is evaluated as the summation of the products as given in the equation. This approach is straightforward and easy to use but only applies to the cases in which all the criteria are additive, and their units are the same. Otherwise, the additive utility assumption is violated, and the results are not consistent.

2.4.2 Weighted Product Method

The Weighted Product Method [119] is the modification of the Weighted Sum Method and is considered to overcome the weakness of WSM. In this method, the total score is evaluated by multiplication rather than addition. According to WPM, to select from various available alternatives, each alternative is compared with the other alternative(s) by finding the product, as shown in Equation 2.8.

$$WPM\left({}^{A}/_{B}\right) == \prod_{j=0}^{m}\left(\frac{a_j}{b_j}\right)^{w_j} \quad (2.8)$$

Where $A, B \epsilon [1, n]$, n is the number of alternatives, m is the number of criteria, a_j is the value of A^{th} alternative with respect to j^{th} criteria, b_j is the value of B^{th} alternative with respect to j^{th} criteria, and w_j is the weight of j^{th} criteria.

The ratio of $WPM\left({}^{A}/_{B}\right) \geq 1$ indicates alternative A is more desirable than alternative B if the criteria are benefiting criteria vice versa in case of cost criteria. The best alternative is the one that is better than or at least equal to all the other alternatives.

WPM's mathematical structure eliminates the units of measure; hence, it is also known as dimensionless analysis and is apt for both one as well as many dimensional problems. Moreover, the approach uses relative values than actual values; hence, more decisive.

2.4.3 Analytical Hierarchical Process

AHP [19][69] is a multi-criteria decision technique that is used to convert the multidimensional problem into a single-dimensional. AHP is hierarchical and linear. In AHP, the problems are initially decomposed into a hierarchy of criteria and alternatives and then arranged in a hierarchical tree. The goal is on the top of the hierarchical tree, whereas alternatives are at the lower levels. This information is then synthesized to determine the relative ranking of alternatives. Both qualitative and quantitative criteria can be compared using informed judgment to derive weights and priorities. The essential feature of AHP is the use of pairwise comparisons to estimate the weights for the criteria and to compare the alternatives with respect to those criteria.

Using pairwise comparison, the relative importance of one criterion over another is expressed in a matrix as A_{n*n} where 'n' is the number of alternatives. Normalized matrix is evaluated, as shown in Equation 2.9.

$$na_{ij} = \frac{a_{ij}}{\sum_{i=0}^{n} e_{ij}} \quad (2.9)$$

For each element na_{ij} of the normalized matrix where a_{ij} is the element of the matrix A_{n*n} which is divided by the sum of e_{ij} that is the element of the corresponding column of A_{n*n} matrix.

Based on the normalized matrix, the criteria weights and eigenvector are evaluated. As proposed by Thomas L. Saaty, the Eigenvector can be used to get a ranking of priorities from the pairwise matrix.

The AHP process provides a logical framework to determine the benefits of each alternative. The AHP has the ability to handle more extensive problems and is ideal for problems that compare performances among the available alternatives. Sensitivity analysis combined with the original AHP is considered as better consistent than the initial approach.

2.4.4 Analytical Network Process

Analytic Network Process [70] is an extension of the Analytic Hierarchy Process. AHP being the basic building block of ANP provides the general framework to deal with decisions without making assumptions about the independence of higher-level elements from the lower-level elements and about the independence of the elements within a level. ANP is the generalized form of AHP and is non-linear, unlike AHP. There is no need to specify levels as in a hierarchy (as in AHP) since ANP uses a network structure. ANP is a useful tool for prediction in cases of complex and networked decision making where the criteria are interdependent. ANP is also valuable for representing a variety of competitors with their surmised interactions and their relative strengths to wield influence in making a decision.

2.4.5 Interpretive Structural Modelling

Interpretive Structural Modelling (ISM) [120] is a methodology that works as a positive means to comprehend an ill-structured and complex model of the system. ISM methodology looks for the inter-relationships between the various organized/structured elements identified as helpful for analysis. ISM methodology

interprets the complex system by the systematic application of graph theory, in an iterative manner. It results in a directed graph of a complex system for a given relative relationship amongst a set of elements. ISM modifies unclear and poorly expressed models of systems to evident, well-defined models that may be useful for numerous purposes. It is a computer-assisted interactive learning process whereby organized models are produced and studied. Structural models, therefore, made portray the structure of a complicated issue, a system or a field of study in patterns carefully designed utilizing graphics and words. However, ISM does not provide any statistically validated models.

2.4.6 Technique for Order Preference by Similarity to Ideal Solution

Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) methodology [121] got introduced as an alternative technique for Multi-Criteria decision-making. Sooner it is becoming one of the preferred variants of MCDM. In this approach, geometric distance is evaluated based on which the best alternative is chosen. Therefore, TOPSIS approach requires the prior evaluation of positive ideal solution (represented as A+) and negative ideal solution (represented as A-) as given in Equation 2.10 and Equation 2.11 respectively.

$$A+= \begin{Bmatrix} \langle \min(t_{ij}) & for\ i = 1\ to\ n \quad and\ j \in jn \rangle, \\ \langle \max(t_{ij}) & for\ i = 1\ to\ n \quad and\ j \in jp \rangle \end{Bmatrix} \tag{2.10}$$

Where

$jn = 1\ to\ m$ criteria associated with the negative impact,

$jp = 1\ to\ m$ criteria associated with the positive impact,

n is the number of alternatives.

$$A-= \begin{Bmatrix} \langle \max(t_{ij}) & for\ i = 1\ to\ n \quad and\ j \in jn \rangle, \\ \langle \min(t_{ij}) & for\ i = 1\ to\ n \quad and\ j \in jp \rangle \end{Bmatrix} \tag{2.11}$$

Where

$jn = 1\ to\ m$ criteria associated with the negative impact,

$jp = 1\ to\ m$ criteria associated with the positive impact,

n is the number of alternatives.

The presumption is that each criterion either increases or decreases the utility, monotonically. Hence, the Euclidean distance approach is used, in general, in TOPSIS for analyzing the proximity of the alternatives to the ideal solution. The alternative having the smallest geometric distance from the positive ideal solution (represented as d_{i+}) and largest geometric distance from the negative ideal solution (represented as d_{i-}) is considered the preferable one.

Distance from Positive Ideal Solution is computed as shown in Equation 2.12.

$$d_{i+} = \sqrt{\sum_{j=1}^{m} \left(t_{ij} - t_{pj}\right)^2} \qquad for\ i = 1\ to\ n$$

Where (2.12)

$$t_{pj} \in A+ \qquad for\ j = 1\ to\ m$$

Distance from Negative Ideal Solution is computed using Equation 2.13.

$$d_{i-} = \sqrt{\sum_{j=1}^{m} \left(t_{ij} - t_{nj}\right)^2} \qquad for\ i = 1\ to\ n$$

Where (2.13)

$$t_{nj} \in A- \qquad for\ j = 1\ to\ m$$

Finally, a similarity to the ideal solution is computed. Generally, the similarity is calculated to negative ideal solution and represented as s_{i-} as shown in Equation 2.14.

$$s_{i-} = \frac{d_{i-}}{(d_{i-} + d_{i+})} \qquad for\ i = 1\ to\ n \qquad (2.14)$$

The range of the similarity is $s_{i-} \in [0, 1]$ that means

s_{i-}= 0 iff the alternative coincides with the negative ideal solution (as d_{i-}=0)

s_{i+}= 1 iff the alternative coincides with the positive ideal solution (as d_{i+}=0)

The ranking of the alternatives is done in the decreasing order of similarity (s_{i-}). However, TOPSIS depends on the computation of the Euclidean distance that does not consider the correlation of the attributes in between. Moreover, it is difficult to compute weight hence maintain the consistency of the selection of the alternative.

The limitations and research gaps identified in the literature so forth are summarized in the following section.

2.5 REVIEW SUMMARY

After the critical look at the available literature, the findings of the research gaps in the literature review are as follows:-

- Need for AOP Software Quality Model:

Every novel software development methodology offers specific techniques that deal with the limitations of the former methodologies. Consequently, it becomes noteworthy to have a quality model in order to understand as well as to measure the impact of a new methodology. After the thorough survey on the existing literature concerning current software quality models, it was found that every model has its own limitations and hence, there is a scope of improvement. Furthermore, the numbers of quality models are very few that capture the effect of aspectization. Considering all, the limitations of the previously available models and the prospective advantages of aspect-oriented software development, there is a need for a quality model that determine the overall effects on the quality attributes as the outcome of aspectization.

- Validation of the proposed AOP Software Quality Model:

The SWOT analysis of the literature review suggests that AHP is quite easy to use as well as scalable. Besides, the hierarchy structure problem can easily be fitted into the

AHP approach. Moreover, the AHP technique is not data intensive. All these features of the AHP technique make it apt for the validation of the proposed quality model.

- Comparison of AOP and OOP on the basis of metrics:

Although there are, various metrics projected by the authors, to determine the estimate of the impact of aspect-orientation on the software development methodology. However, relatively less work is done in the field of relative comparison between an aspect-oriented software system and their object-oriented equivalent.

- AO System Coupling Metric:

It was identified that although metrics exist for assessing the coupling of aspect-oriented systems, they are at the basic levels of fields, methods, classes, or are aspects as standalone entities. Only a few metrics exist for the measurement of software quality attributes at a higher level of abstraction in AO systems. Hence, there is a need for one metric for complete Aspect-oriented Software System Coupling.

- Supportability Metric:

Though, software product supportability is an essential feature for improving quality. But there is no proper way to measure the Supportability characteristic of the software. Hence, there is a need for a metric for measuring supportability using the obtrusive data collection approach that may be used to early diagnosis of the specific software weakness in terms of usage easiness and the acceptance of the customer.

- Extensibility Framework:

Even if all the researchers agreed on the point that Aspect-oriented approach could contribute positively to extending the software design and code dynamically. However, still, there is a vacancy for a formal framework for evaluating the extensibility of the software. Hence there is a need for formal proposal and validation of extensibility metric for Aspect-oriented Software System. Such a framework will help software developers in selecting software that can be easily extensible.

With a view to resolve the aforementioned issues and to fill in the research gaps, various optimized solutions along with their validation are presented in the subsequent Chapters.

Considering the various limitations identified with regards to existing quality models as listed in this Chapter, and to incorporate the prospective advantages of aspect-oriented software development, an enhanced software quality model is proposed by augmenting a few new characteristics to the existing ISO25010 quality model. The details of the proposed software quality model are presented in the next Chapter III.

CHAPTER III

QUALITY MODEL FOR ASPECT-ORIENTED SYSTEMS

3.1 INTRODUCTION

The quality of the system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and hence provides value. These stated and implied needs are represented through quality models. A quality model is a hierarchical decomposition that categories the product quality into a set of characteristics, which are further divided into sub-characteristics [45], as demonstrated in Figure 3.1.

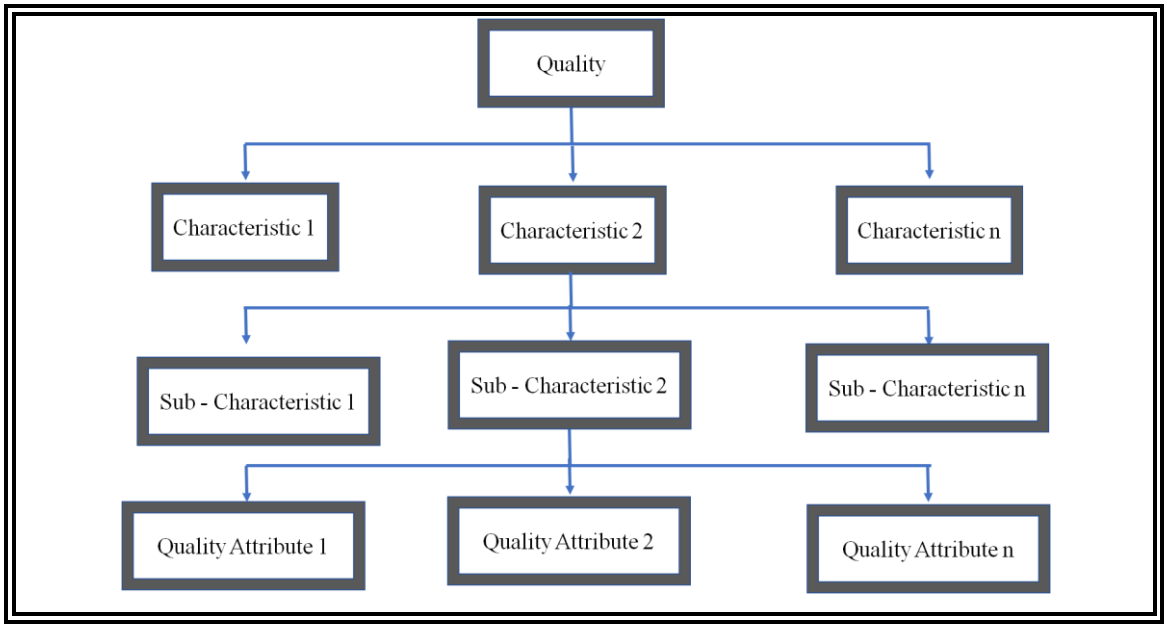


Figure 3.1: Hierarchical Software Quality Model

Quality of a particular characteristic can be measured by measuring quality attribute or attributes associated with that characteristic or set of sub-characteristics that compose the characteristic. That is, it is possible to measure directly or by computationally combining the collection of quality attributes [30][45].

In Chapter II, the different software quality models are reviewed, and it is evident that each of the models mentioned has its limitations. Moreover, even though a number of quality models have been proposed for object-oriented software system but only a few

of them are capturing the advantages of aspect-orientation. Considering the limitations of the previously available models, and the prospective benefits of aspect-oriented software development, the missing characteristics that are relevant for the modern-day software are augmented to enhance the existing ISO25010 quality model are identified. Thereafter a novel quality model for Aspect-Oriented software quality model on the concept of ISO 25010 is proposed. The detail discussion of the proposed quality model framework is given in the following sections.

3.2 A NOVEL SOFTWARE QUALITY MODEL FOR ASPECT ORIENTED SYSTEM

A lot of hard work and labor is invested in delivering a quality product. The primary aim of any developer is to provide the product with all the three rights; that is, right product, at the right time with the right functionalities. As every customer expects that the best outcome is delivered to them, hence the onus to ensure that lies with the developers and testers.

In this section, a novel software product quality model is proposed to incorporate the modern-day software features. The proposed software quality model is inspired from ISO/IEC 25010, which has recently replaced the ISO/IEC 9126 product quality model. The proposed model is depicted in Figure 3.2.

Once the software is deployed, it needs to fulfill the expected level of performance. It needs to be reliable, available, and even scalable if required. If the coding lacks these run-time quality attributes, then it is considered as useless regardless of how proper the coding is in terms of the user interface or features. Hence these quality attributes are needed to be designed and coded carefully into the system to make it usable. All these are the run time attributes of the software product or project that enhances the confidence in the customer/end-user, which ultimately boost the software quality.

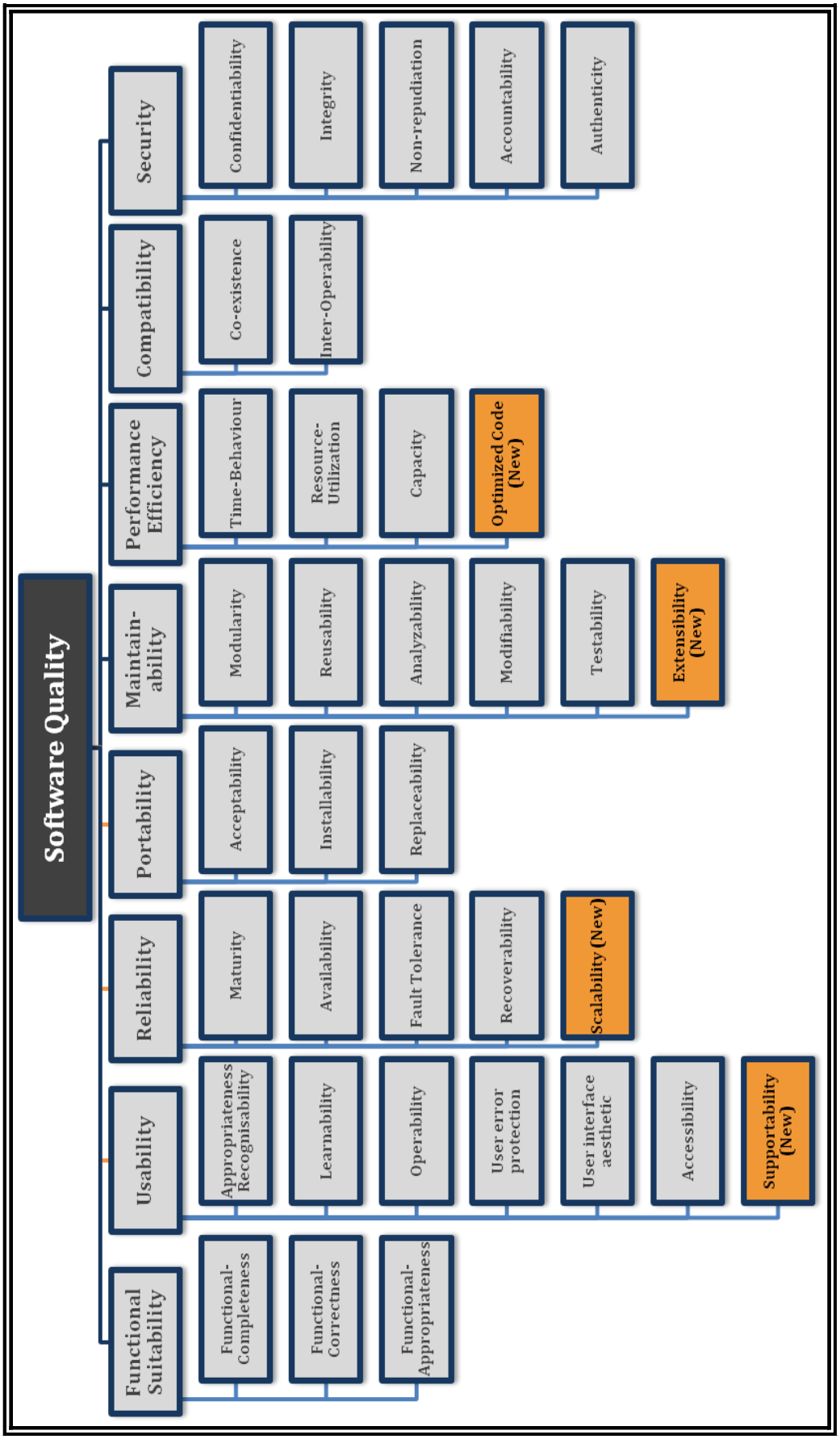


Figure 3.2: New Proposed Software Quality Model

In present times, software systems are highly complex systems that should work in real-time. While developing software applications, building reliable systems become difficult, but they are necessary as their failure may result in tremendous economic cost or loss of life and even complete mission failure. Unreliable systems are liable to be rejected by the customers as they may cause a considerable information loss. Reliability invokes a sense of trust and has become an integral expectation by the customer. Planned software engineering significantly reduces the risk of the development of unreliable software system. Reliability characteristic of the system is composed of hardware reliability and software reliability. While developing software, the concern is on software reliability. Software Reliability, in general, is understood as the probability that the system will continue to perform its intended functions for a defined period of time under a specified set of the environment conditions.

Additionally, users perceive software in a way that represents its functions. Even minute details in the user interface affect the impression of the overall software product on the customers. One of the fundamental reasons for the aversion among users of the interactive software is it being unfriendly, sophisticated, and low-quality user-interfaces. Software with poor user interface design not only waste the time of its users but also disappoints them and adds to their frustration. Incorporating affable 'ease of use' in software is a tedious task in itself. Ideally, the source code should be written in a manner that reduces the effort required to comprehend its behavior. Most source code programming style guides often strain on enhancing the readability of the software. They focus on following the language-specific conventions in order to reduce the outlay of source code maintenance [51][52][53]. Traditionally, with the focus entirely on the functionality of the software, the efforts were mainly devoted to improving the efficiency of the software in terms of time, memory, and cost [54]. However, in this day and age, the extent to which the customer can effectively and efficiently use a software product with reasonable satisfaction has a significant role in gaining acceptance for the software from prospective customers. This change in the current scenario has led to an emphasis on the usability features of the software. Consequently, Usability has emerged as one of the critical elements of software quality. It has been considered a vital quality characteristic by various quality models [55][56]. It is a relative term, which cannot be defined in an absolute sense; instead, it can be defined only in a particular context. It is dependent on what are the

specifications of the product to be developed? Who will be the specified users? What specified goals are to be achieved by developing that software product? Moreover, what will be the specified context of the usage of that software? [57]

Considering all the transformation in the present-day state of affairs, four relevant characteristics are identified to be missing in the latest software quality model, ISO 25010, and hence augmented in the proposed software quality model, as follows:

I. Extensibility in Software Maintainability
II. Scalability in Reliability
III. Supportability in Usability
IV. Optimized code in Performance Efficiency

The detailed discussion of the relevance of the proposed characteristics is presented in the subsequent sections. In the next section, the details of the extensibility attribute in software maintainability are given.

3.3 EXTENSIBILITY

Extensibility can be defined as a systematic measure of the ability to extend or enhance the current software and the degree of effort required to implement the extension while minimizing impact to existing system functions. Software development process leads to the delivery of high-quality software product that satisfies the user requirements. Accordingly, the developed software product should eventually be able to change or evolve after delivery.

Change is pervasive in software development. For any software product to be maintainable, it is the easiness with which it incorporates any change. The change can be corrective, adaptive, perfective (enhancement) or preventive [45].

- *Corrective change* is to correct defects in the software.
- *Adaptive change* results in the modification in the software to accommodate changes to its external environment.
- *Perfective change* extends the software beyond its original functional requirement.

- *Preventive change* is to make computer programs easily corrective, adaptive, and perfective.

The mapping of these different types of changes on to the related maintainability sub-characteristics is highlighted in Table 3.1.

Table 3.1: Type of Change concerning maintainability sub-characteristics

Type of Change	Maintainability sub-characteristics
Corrective	Modifiability
Adaptive	Reusability
Perfective	NONE
Preventive	Modularity, analyzability, modifiability

As shown in Table 3.1, there is no sub-characteristic related to perfective change; and hence, there is a need for an added new sub-characteristic named ***Extensibility***.

3.3.1 Software Extensibility in Existing Quality Models

Different quality models have been studied in an attempted to identify the importance of software extensibility aspect concerning maintainability.

Jim Mac Call [23] identified expandability under the flexibility criteria for the product revision perspective that defines the ability of the software product to go undergo changes, including error corrections and system adaptations.

Barry W. Boehm [25][26] aggregated the augmentability feature within the scope of modifiability characteristics of the defined hierarchical software quality model. That is, augmentability is considered as necessary characteristics for modifiability. As per the software quality model defined, the augmentability is described as the extent to which the code could comfortably accommodate expansion in component computable functions in data storage requirements.

In Robert Grady's FURPS (Functionality, Usability, Reliability, Performance, and Supportability) Quality Model [58], the supportability characteristics include extensibility as an explicit Quality Aspect in concern to supportability.

ISO 9126 quality model [27] does not explicitly address extensibility characteristics to represent future growth of the software. However, the argument may be that the enhancement of new features, type of change, is embedded within the kinds of modifications defined in the quality model. That is, corrections, improvement, or adoptions of the software to changes in the environment, requirements, and functional specifications. But, then although changeability/ modifiability are separately taken as sub-characteristic in maintainability characteristic and adaptability sub-characteristic is separately taken as sub-characteristic in portability characteristic. However, there is no separate sub-characteristic of extensibility for improvements.

ISO 25010 [30] made an amendment in ISO 9126 and reusability is added as sub-characteristic to maintainability, but it still misses the extensibility feature as it tells how easily code can be extended to added further functionality/changes, etc.

Table 3.2: Extensibility Attribute Coverage in Quality Models

Extensibility coverage in Quality Models					
Quality Model	**McCall**	**Boehm**	**FURPS**	**ISO 9126**	**ISO 25010**
Extensibility	Explicit	Implicit	Explicit	Implicit	Implicit

The coverage of extensibility attribute in the software quality models is summarized in Table 3.2, indicating that extensibility sub-characteristic is partially, implicitly, explicitly addressed, or not addressed at all in quality models.

Although the attributes of changeability/ modifiability, portability, adaptability are addressed explicitly in most of the quality models but extensibility is explicitly addressed only in McCall quality model [23] and FURPS model [58] and not in the other software quality models.

AOSD aims at encapsulating crosscutting concerns through a new construct class like Aspect and localize the change. Hence, it encapsulates the behavior affecting multiple classes into the reusable module. Moreover, the software maintenance is not only to correct faults but also to extend the software for implementing new or changed user requirements, which concerns functional enhancement; hence, a modified maintainability model is proposed for AOSD.

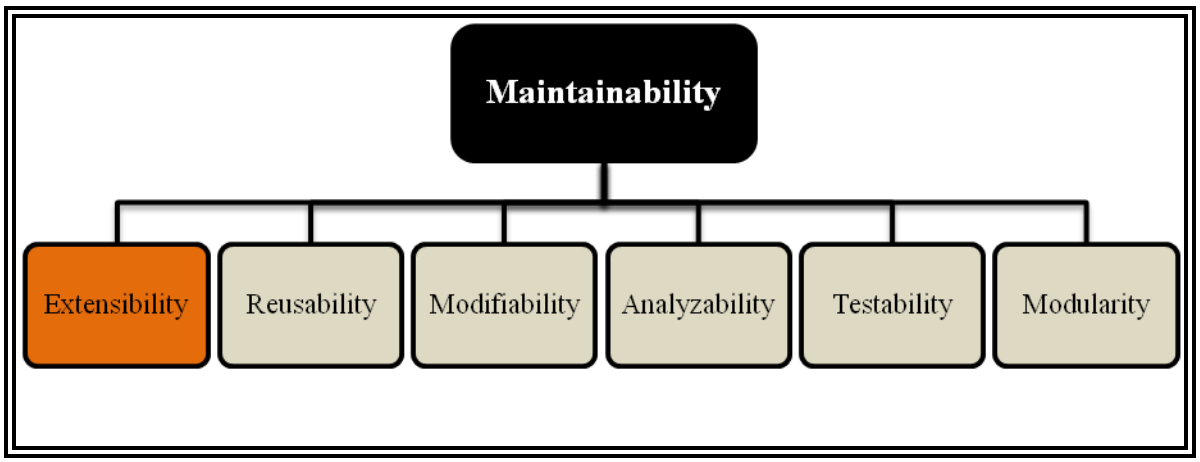


Figure 3.3: Maintainability with the sub-characteristics

In order to provide complete coverage of the type of change, the inclusion of extensibility as a sub-characteristic under the maintainability quality characteristic is recommended. As every proposed model requires evaluation hence, in order to evaluate the proposed maintainability model [108] Analytical Hierarchy Process, (AHP) is used as an approach and the details are compiled in Chapter IV.

The details of the second proposed quality attribute, Scalability in the software reliability model is given in the following section.

3.4 SCALABILITY

Scalability is the capability of the system to either handle an increase in load without affecting the performance of the system or the capability to be readily enlarged.The scalability of software is vital for growing business. The key to scalability is that the software grows along with the increased usage.

Once the software is deployed, it needs to fulfill the expected level of performance. It needs to be reliable, available, and even scalable if required. All these are the run time attributes of the software product or project that enhances the confidence in the customer/ end-user which ultimately improves the software quality.

Reliability of software product or project is widely accepted as an essential characteristic of software quality. For a software system to be reliable, it should be available irrespective of faults or failures or even application crashes. That means a reliable system should not lose its availability even in most failure situations or even under heavy load conditions.

What if the demand is increased for the software functions? Users will face increased response time and longer completion time. The system cannot queue the excess load and process it at the reduced load period. How such a system considered as reliable? For a software product or project to be truly reliable, it should continue to operate even under heavy load conditions.

As per International Standard Organisation, software reliability is the degree to which it performs its intended functions under specified conditions for a specific period of time. And scalable software can provide support to the increased amount of data or can cater to the increased number of users without degrading the performance drastically. So, Scalability is an essential characteristic of a system to be reliable.

3.4.1 Software Reliability in Existing Quality Models

Reliability characterized by specific attributes as highlighted by various quality models [59] is investigated as follows:

Jim Mc Call quality model [23] identified reliability as a product operation quality attribute that influences the essential operation of the software.

Barry W. Boehm's hierarchical software quality model [25] identified reliability as a quality factor associated with the As-is general utility or primary use. According to this model, code possesses the reliability characteristic to the extent that it can be expected to perform its intended functions.

FURPS quality model [58] considered reliability as the non-functional quality requirement. According to this model, reliability may include frequency and severity of the failure, recovery to failure, and time among failures.

ISO/IEC9126 standard [27] for quality considered reliability as the capability of the software product/project for maintaining a specified level of performance when used under specified conditions. The sub-characteristics identified are maturity, fault tolerance, and recoverability.

Ghezzi software quality model [60], considered reliability as a desirable external quality attribute which can be achieved by the improvement in the internal quality attributes of the software.

Khosravi proposed a software quality model [61] for accessing the quality of software implemented using design patterns. It considered scalability as a quality characteristic defined by the level of performance and application performance. It is considered to add software elegance.

Khomh [62] proposed DEQUALITE (Design Enhanced Quality Evaluation) to build a quality model for object-oriented systems. It considered scalability as a quality attribute under attributes related to runtime.

ISO 25010 software quality model [30] defined reliability as the degree possessed by the software product/project to perform specified functions under specified conditions for the specified time period. As an upgradation of the previous ISO/IEC 9126 software quality model, the availability characteristic for the software product was added as a sub-characteristic for software reliability quality characteristic. The reliability sub-characteristics are, namely maturity, fault tolerance, availability, and recoverability.

Although one attributes are included in the reliability characteristic but the missing point in ISO25010 model concerning reliability is that while estimating the reliability of software, no weight age is given to how scalable the code is and how much efficiently it handles the increased load. Software reliability goes hand in hand with the availability and scalability. An unreliable software system is also un-scalable.

Considering all, a new hierarchical software reliability model is proposed.

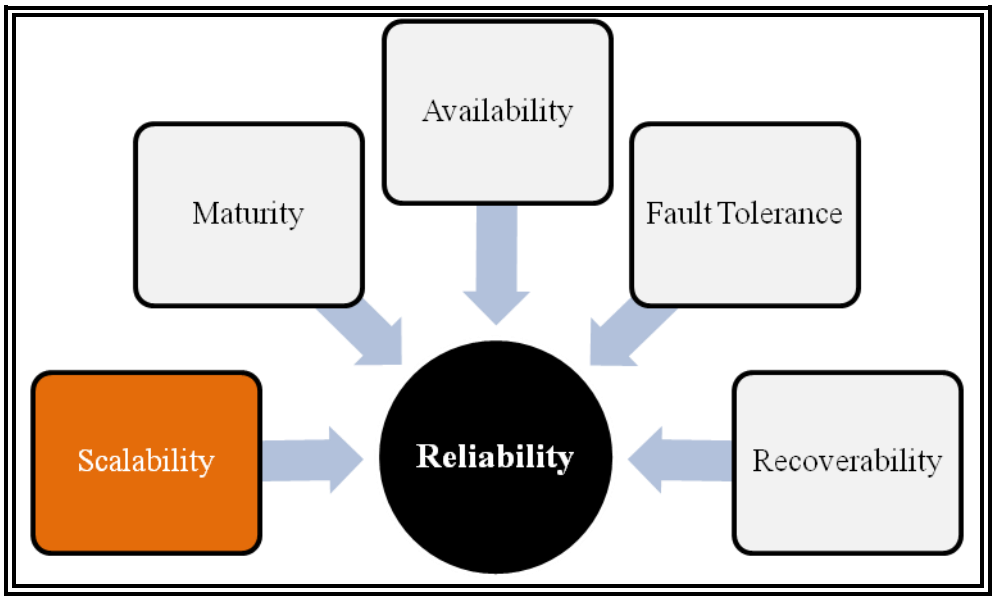


Figure 3.4: Reliability with the sub-characteristics

The proposed reliability model is an extended model with scalability as an additional sub-characteristic under reliability along with maturity, availability, fault tolerance, and recoverability as given in Figure 3.4. To verify the consistency of the proposed model, an assessment is made by the application of one of the MCDM (multi-criteria decision making) approach, namely the AHP (analytical hierarchy process). Results concluded confirm the proposed reliability model to be consistent and may further be used for computing the overall quality of software product or project and are compiled in Chapter IV.

The detailed description of the proposed supportability attribute in the usability model is presented in the next section.

3.5 SUPPORTABILITY

Supportability is the effectiveness of the system to afford information that helps identify and resolve issues when it fails to work correctly. From an individual point of view, source code can be written in a way that affects the effort required to comprehend its behavior. Software is written to accomplish some specified objectives for specified customers. In order to be readily acceptable, software needs to be easily

understandable and usable. Clear, complete, and good quality supporting material provided along the code enhances the chances of ready acceptability by the end-user. Many source code programming style guides, which often emphasizes readability and usually language-specific conventions intend at reducing the cost of source code maintenance.

Software Usability as defined by IEEE "is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component."

Although usability is a widely accepted term for the evaluation of software quality, but there is a lack of consistency in the definitions of usability given by various standards. Even the given definitions miss out on the vital aspects affecting the usability of the software product. In absentia of clear guidelines about how usability is related to its sub-factors and the criteria, ad hoc implementation of usability has been done by the developers [63]. Usability valuation in the model-driven development procedure is still an area where further research works are desirable [64].

ISO also updated the previous model ISO9126 and released ISO25010 software quality model. Definition of Usability factor given by the ISO 9126 model is also modified in ISO25010 model. Its new description included terms like specified users and specified goals, along with the specified context of use. It drew attention to three sub-goals. First, effectiveness that defines the potential to produce the preferred outcome. Second, efficiency that determines the amount of time, effort, and cost used during the accomplishment of the task. Moreover, third, the satisfaction that defines the likability of the software to use. Appropriateness, recognizability, learnability, operability, user error protection, user interface aesthetics, and accessibility are acknowledged as the sub-characteristics by ISO25010 model for considering user interface design with good usability [63].

Although many sub-characteristics are defined for usability in ISO 25010, one crucial characteristic is missing, and that is Supportability. The features for supportability may include:-

- Documentation
- Trouble Shooting tools

- Help menu
- Event logging/Tracing Code etc. as shown in Figure 3.5.

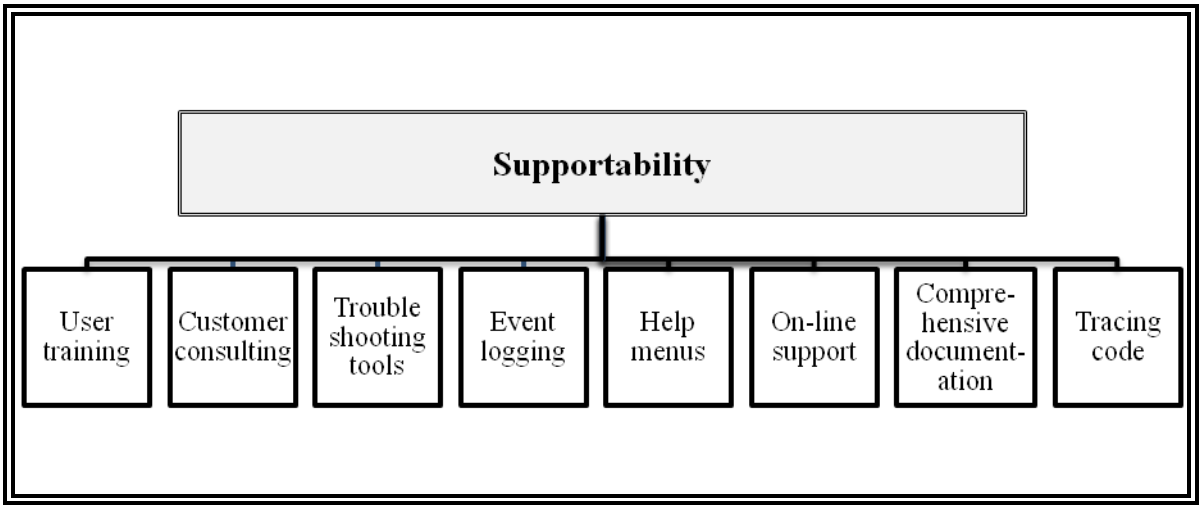


Figure 3.5: Supportability and its features

Software having proper documentation supported by help menus and troubleshooting tools along with traceability; makes the software more effective, efficient, and satisfying than before; hence more usable.

3.5.1 Usability in Existing Quality Models

Usability is one of the desirable as well as a vital software product attribute. Hence, all the existing software quality models incorporate usability as an integral attribute [65][66][67].

Mc Call software quality model [23] comprises of eleven quality factors; one of these factors was usability. These factors were further categorized into quality criteria's. Usability was categorized into communicativeness, operability, and training. No metrics were defined for measuring usability.

Barry Boehm [25] and et al. like Mc Call model [23], also comprises of quality factors, which have been further categorized into quality criteria's and metrics. However, unlike Mc Call, usability quality attribute was not considered as a quality factor; instead, it was considered as an outcome of portability and maintainability quality factors.

Grady's FURPS software quality model [58] is a user-centric model and this model values more on user requirements rather than the developer's requirement. Usability being a user-centric requirement is essential for trouble-free acceptance of the developed software. Hence, it was considered as one of the crucial features for quality. It comprised of various sub-factors like user documentation, human factor, online and context-sensitive help, steadiness in the appropriate interface, wizards, and training materials.

Ghezzi software quality model [60] was based on the structural qualities of the software. According to this model, improvising the structural quality of the software implicitly improves the overall quality. It also considered usability as one of the factors affecting the software quality.

ISO 9126 model [27] defined usability as the ability of the developed software product to be understandable, learnable, usable, and attractive to the end-user. It was limited by the usage of the software under specified conditions. It comprised of five sub-quality components, namely understandability, learnability, operability, attractiveness, and usability compliance.

Dromey defined a generic and dynamic software quality model [68] which was supposed to be compatible with a variety of software. It was a two-layer model. Higher layer composed of product properties that affect the quality and the lower layer was composed of quality attributes. Usability was considered as a quality attribute under the descriptive property of the quality model.

Kumar et al. [50] proposed a software quality model specifically for Aspect-Oriented Software Development, named Aspect-Oriented Software Quality Model (AOSQUAMO Model). This model was inspired by the ISO 9126 software quality model standard. Although none of the existing attributes of quality in the ISO 9126 model were changed/moved/deleted, but, four new attributes were added. Usability remained one of the quality factors in AOSQUAMO model as in the ISO 9126 model.

In International Standards Organization latest quality standard model ISO 25010 [30] although various attributes were revised, Usability retained its existence as a quality factor in the quality model.

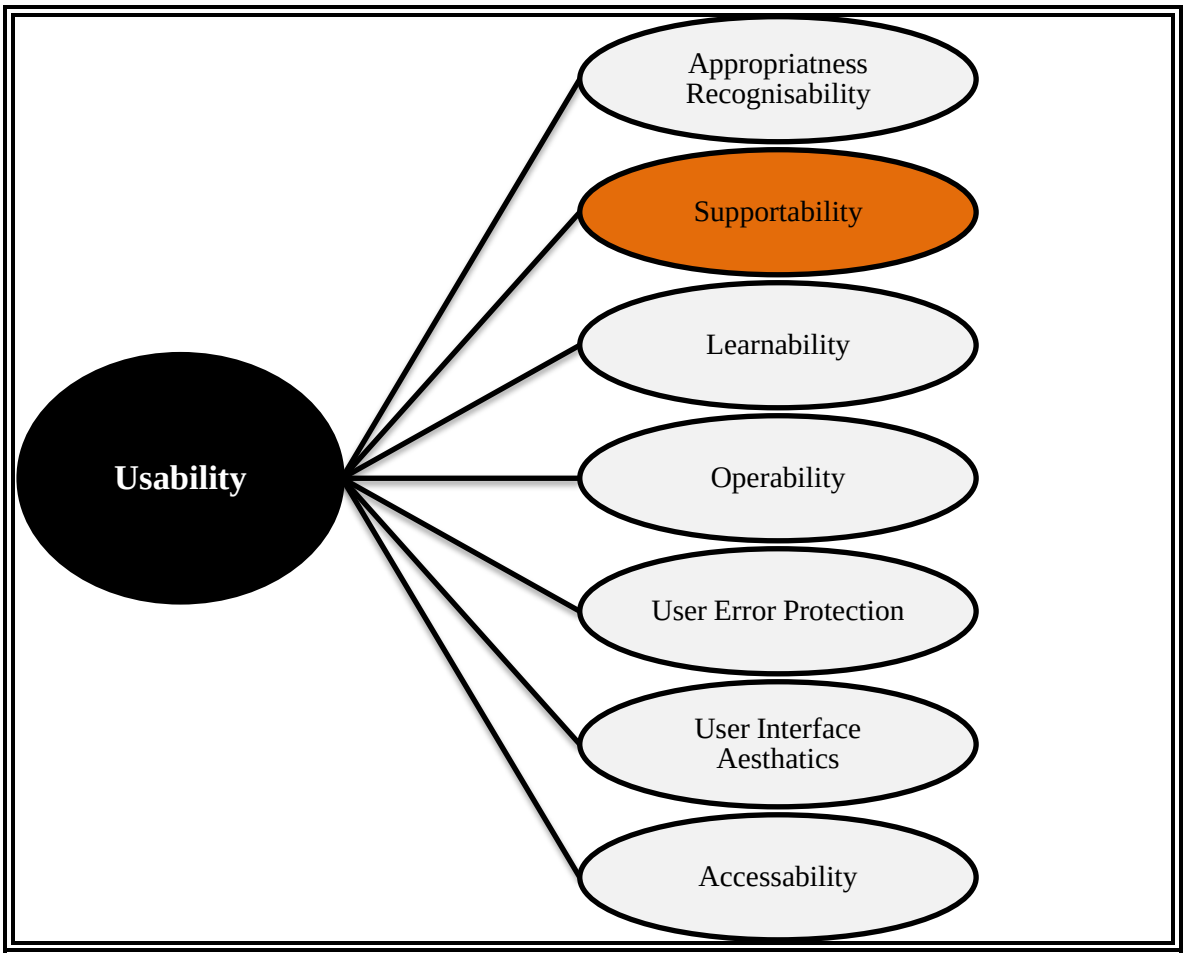


Figure 3.6: Supportability with the sub-characteristics

Supportability characteristics of earlier software product constituted mainly of maintenance and fixing of errors/issues, but now it encompasses a broad and extensive range of elements including a telephonic or online support system and customer consulting. Even though user-error protection and accessibility have been added as the new sub-characteristics of usability under ISO25010 model, but still one fundamental characteristic that widely affects the software usage and acceptance is missing, and that is Supportability. Considering all the relevant factors, an enhanced hierarchical usability model is proposed with supportability as one of the sub-characteristics as shown in Figure 3.6. The relevance of the proposed model has been assessed using Analytical Hierarchical Process (AHP), a technique of multi-criteria decision-making, with the involvement of a team of participants working in different platforms in different companies and the details, are given in Chapter IV.

The detailed discussion of the proposed optimized code attribute in the performance efficiency model is described in the subsequent section.

3.6 OPTIMIZED CODE

A well-written code can significantly reduce the effort required in further phases of the software development life cycle like testing and maintenance. Use of solid coding techniques and good programming practices for creating high quality, optimized code plays a vital role in software quality and performance.

Software quality should not only be able to meet the customer requirements but should exceed it. Customers not only mean the external customers but the internal ones too. Code that is written by consistently applying well coding standard and proper coding techniques is not only optimized in terms of time, effort, cost (resources) but also is more comfortable to comprehend and maintain.

Performance efficiency is the performance relative to the extent of resources used under stated conditions. It is an indication of the responsiveness of a system to execute specified actions in a given time interval and is considered as one of the vital software quality characteristics. If performance efficiency is improved, then it will certainly have a positive effect on software quality.

The ***optimized code*** has a positive effect on performance in terms of less response time, increased throughput, reduced memory consumption, and reduced network bandwidth consumptions.

3.6.1 Performance Efficiency in Existing Quality Models

Various software quality models have been reviewed to understand the perspective for taking the performance efficiency as a characteristic for defining the quality.

Jim McCall [23] considered efficiency as one of the quality factors under product operations. It defined one or more quality criteria for each quality factor in order to assess the overall quality of software product. According to Mc Call quality model, the quality criteria for efficiency are execution efficiency and storage efficiency.

Barry Boehm software quality model [25] identified efficiency as a quality attribute under As-is utility. According to Boehm quality model, the factors that affect efficiency are accountability, device efficiency, and accessibility.

ISO 9126 software quality model [27] identified efficiency as one of the quality characteristics and specifies three quality attributes that affect the efficiency of software are time behavior, resource behavior, and efficiency compliance.

In Kumar et al. [50] extended ISO/IEC 9126 quality model, AOSQUAMO (Aspect-Oriented Software Quality) model, added code- reducibility as a sub-characteristic under efficiency quality characteristic. Hence, the quality attributes that affect the efficiency according to AOSQUAMO model are time behavior, resource utilization, and code reducibility.

In ISO/IEC 25010, the problems related to performance efficiency are addressed, and capacity is added as a sub-characteristic alongside the previously exiting characteristics of time behavior and resource utilization [30]. And capacity is expressed as the extent to which the software product maps to the specified requirements.

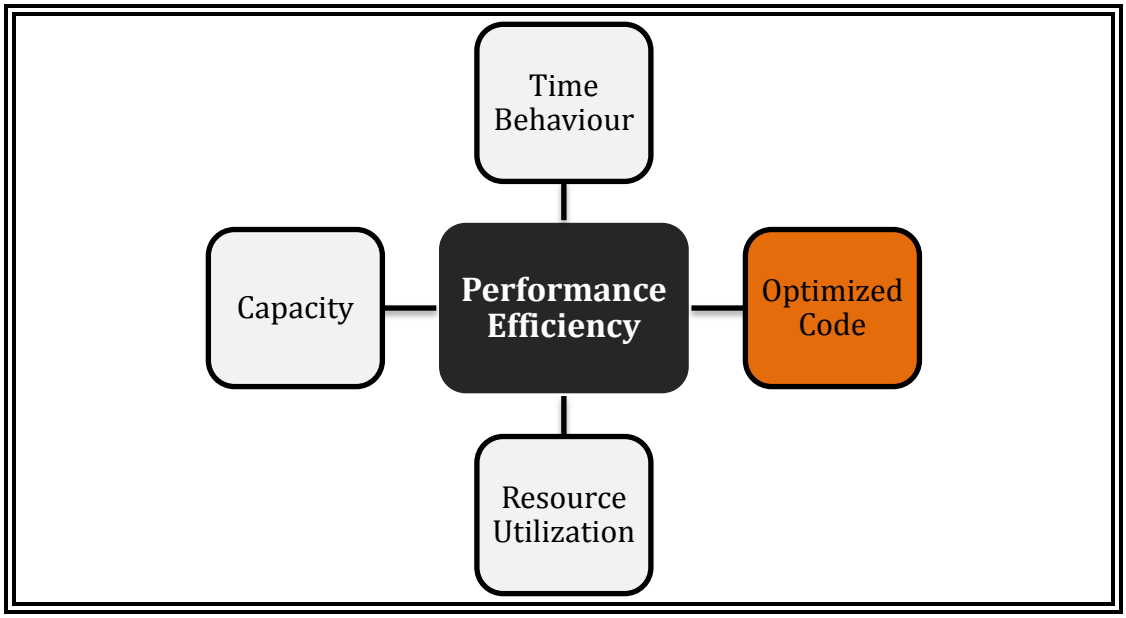


Figure 3.7: Performance Efficiency with the sub-characteristics

Although in ISO 25010, the gaps related to performance efficiency were addressed but still, one area is left untouched, so, optimized code as a sub-characteristic for Performance Efficiency is proposed as depicted in Figure 3.7.

The validation of the four newly added attributes is done using Analytical Hierarchy Process (AHP) technique of Multicriteria Decision Method, and the classified details are given in the next Chapter IV.

CHAPTER IV

VALIDATION OF THE PROPOSED MODEL

4.1 ANALYTICAL HIERARCHICAL PROCESS

The procedure followed by the analytical hierarchical process technique [19][69] is comprised of the following steps:-

- Step 1: Define the problem and state the goal or objective.
- Step 2: Define the various characteristics affecting that objective and make the hierarchical structure. Problems are bifurcated into a hierarchy of criteria and alternatives, as shown in Figure 4.1.

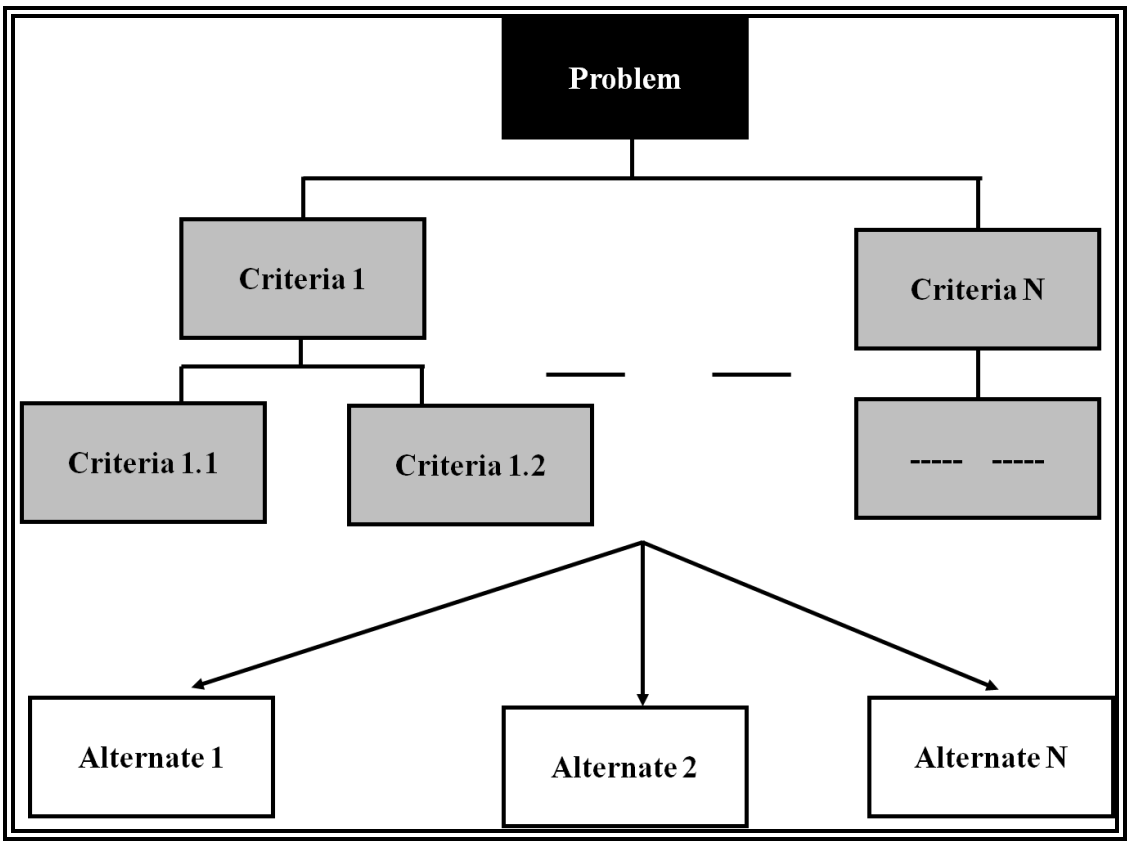


Figure 4.1: Hierarchy of Criteria and Alternatives

➢ Step 3: A survey is conducted using a data collection form consisting of pairwise comparisons for filling relative weight of criteria's C_1 to C_n in the scale of 1 to 9 according to Thomas L. Saaty [19] as presented in Table 4.1.

Table 4.1: Scale of Relative Importance

Intensity of Relative Importance	Definition
1	**Equal Importance**
2	**Very Weak Importance**
3	**Weak Importance**
4	**Moderate Importance**
5	**Strong Importance**
6	**Strong Plus Importance**
7	**Very Strong Importance**
8	**Very Strong Plus Importance**
9	**Extreme Importance**
Reciprocals(1/2 to 1/9)	**For vice-versa comparisons. That is if the relative importance of C_i to C_j is 5 then C_j to Ci is 1/5**

Use paired comparisons of each criterion concerning each other after structuring the characteristics into levels and sub-levels. The information is then synthesized to determine the relative ranking of alternatives. Both quantitative and qualitative criteria can be compared using informed judgments to derive weight and priorities. The relative importance of one criterion over another can be expressed using pairwise comparisons, as demonstrated in Table 4.2.

Table 4.2: Sample Matrix for pairwise comparison

	C1	C2	C3	C4	C5
C1	**w1/w1**	**w1/w2**	**...**	**...**	**w1/wn**
C2	**w2/w1**	**w2/w2**	**...**	**...**	**w2/wn**
C3	**...**	**...**	**...**	**...**	**...**
C4	**...**	**...**	**...**	**...**	**...**
C5	**wn/w1**	**wn/w2**	**...**	**...**	**wn/wn**

- Step 4: Calculated the selected characteristics in relevance to the defined objective known as Eigenvector.
- Step 5: Evaluate the consistency
  - Step 5.1: Evaluate the consistency index using the eigenvector as given in Equation 4.1.

$$CI = \frac{(\lambda_{max} - n)}{n - 1} \quad (4.1)$$

  - Step 5.2: Evaluate the consistency ratio to ensure and verify the consistency of the comparison matrix as shown in Equation 4.2.

$$CR = \frac{CI}{RI} \quad (4.2)$$

According to Saaty, for 3x3 matrix if CR > 0.05 ; for 4x4 matrix if CR > 0.08 ; and for all the larger matrixes if CR > 0.1 then the input set for judgment is too inconsistent.

RI depends on the number of alternatives (in case of alternatives comparison) or criteria (in case of criteria comparison) being compared; hence, it varies depending upon the order of the matrix. A scale for RI is a known random CI obtained from a large number of simulation runs and is predefined.

- Step 6: After the consistency verification, various characteristics are evaluated according to their weight, and finally, we get the rank of characteristics, and the final choice is made.

The validation of the proposed software maintainability model is done, and the details are given in the subsequent section.

4.2 SOFTWARE EXTENSIBILITY AS A SUB-CHARACTERISTIC IN SOFTWARE MAINTAINABILITY

According to IEEE, the Software Maintenance is "the process of modifying a software system or component after delivery to rectify faults, improve performance or other attributes, or adapt to the changed environment" [1][2]. The most common perception about maintenance is that it merely involves fixing defects. However, the fact is that software maintenance is an extensive activity that not only covers error correction but includes enhancements of the capabilities, adaptations to the new environments and optimization. Hence, Software Maintenance forms an integral part of defining the software quality model. Different researchers have proposed different software quality models to help to measure the quality of software products. In the research, various renowned software quality models are reviewed and the modified software quality model is proposed. However, the empirical validation is yet to be reported for aspect-oriented software [2][8].

In this section, the related work and the proposed model has been discussed in context to the maintainability of aspect-oriented software.

The manner in which software maintainability has been addressed in the software quality models is given as follows:-

Barry Boehm identified 7 quality attributes (or factors) according to the 3 primary uses of the software that could affect the quality of the software product. The underlying focus of Boehm was on maintainability [25].

ISO 9126 product quality model enumerates five quality attributes that affect the maintainability of software are analyzability, changeability, stability, testability, and maintainability compliance [27].

Kumar et al. proposed the AOP based software quality model, namely AOSQUAMO (Aspect-Oriented Software Quality) model, which is consequential from the ISO/IEC 9126 quality model. The quality attributes that affect the maintainability according to AOSQUAMO model are analyzability, changeability, stability, testability, and modularity [50].

According to ISO25010 model, the quality attributes that affect maintainability are updated to modularity, reusability, analyzability, modifiability, and testability [30]. Although many problems of ISO 9126 concerning maintainability have been addressed in ISO 25010 but few areas are left.

In order to analyze the problems in the area of maintenance and enhancement of application software, a survey was conducted by Lientz, Swanson, and Tompkins. In 1978, they published the results in Communications of ACM, indicating that perfective maintenance is the most significant area of effort. Moreover, within this category, user enhancements and extensions account for two-thirds of the total effort. Although the survey was conducted 35 years ago, but the results of this survey are one of the most widely cited papers on software maintenance and continues to be quoted regularly [71]. Since the majority of effort and resources and spent during the maintenance phase; hence, it dramatically affects the cost of the software product.

The software maintenance is not only to correct faults but also to extend the software for implementing new or changed user requirements which are related to functional enhancement hence a new maintainability model is proposed for AOSD with extensibility as sub characteristic as given in Figure 3.2 of Chapter III.

As every proposed model requires validation; hence, in order to validate the proposed maintainability model, the Analytical Hierarchy Process (AHP) as an MCDM approach is used, and the details are as follows.

4.2.1 Validation

Step1: Maintainability is decomposed into a hierarchy of sub-characteristics extensibility as Criteria C_1, reusability as Criteria C_2, modifiability as Criteria C_3, analyzability as Criteria C_4, testability as Criteria C_5, and modularity as Criteria C_6, as

shown in Figure 4.2. Java Software and AspectJ Software are taken as the two alternatives to choose from.

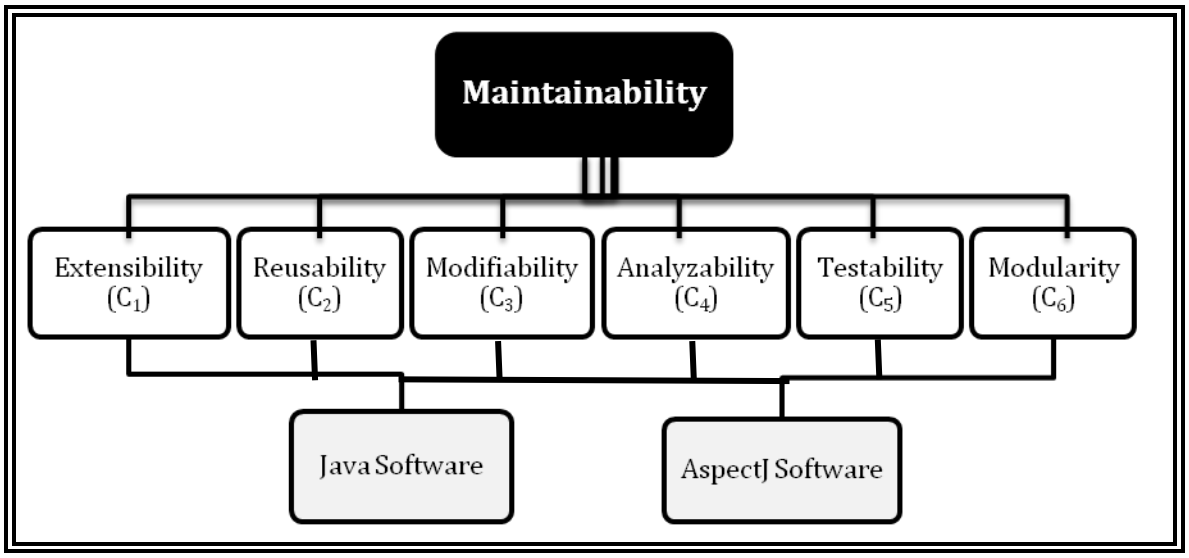


Figure 4.2: Proposed Maintainability Model

Step2: An industrial survey has been conducted using a form (as given in Appendix A Figure A1.1, Figure A1.2 and Figure A1.3) consisting of 15 comparisons for filling pairwise relative weight of characteristics C1 to C6 in the range of 1 to 9 is provided to each individual. Mean of collected samples of pairwise relative weights is calculated and allocated to sub-characteristics for further evaluation, as shown in Table 4.3.

Table 4.3: Matrix M for weights allocation to sub-characteristics for maintainability

		C1	C2	C3	C4	C5	C6
M=	**C1**	1.0000	2.0175	1.4213	1.9313	2.2125	2.1300
	C2	0.4957	1.0000	2.0450	2.5800	2.0288	1.9613
	C3	0.7036	0.4890	1.0000	1.9425	1.8538	2.1063
	C4	0.5178	0.3876	0.5148	1.0000	3.3150	2.8888
	C5	0.4520	0.4929	0.5394	0.3017	1.0000	2.9900
	C6	0.4695	0.5099	0.4748	0.3462	0.3344	1.0000

Step 3: To determine the consistency of the allocated weights, eigenvector, and eigenvalue are calculated.

In order to compute the eigenvector, the pairwise Matrix is raised to the powers that are successively squared each time. After that row sums are calculated and normalized.

First Iteration: Resulting matrix after squaring the M Matrix is given in Table 4.4.

Table 4.4: Matrix for M^2 after 1st Iteration

		C1	C2	C3	C4	C5	C6
M^2=	C1	6	7.65515	10.1673	13.233191	18.26711	23.4046
	C2	5.603846	6	8.148207	11.380578	18.15371	22.80448
	C3	4.482118	5.1381	6	7.793775	13.4001	17.82434
	C4	4.444458	5.178513	5.717939	6	10.48242	18.63674
	C5	3.087787	3.80294	4.304153	4.830841	6	9.917066
	C6	1.856161	2.498136	3.018136	3.937662	4.769718	6

The row sums and the eigenvector calculated on Matrix M are shown in Table 4.5. The process is iterated until the eigenvector solution does not change from the previous iteration.

Table 4.5: Matrix for row sum and eigenvector for Matrix M^2

	Row Sum		Eigenvector
	78.7274		0.254
	72.0908		0.2326
	54.6384		0.1763
	50.4601		0.1628
	31.9428		0.1031
	22.0798		0.0712
Row Total	309.9393	Total	1.0000

Second Iteration: The M^2 matrix is squared, and the resulting M^4 Matrix is shown in Table 4.6

Table 4.6: Matrix for M^4 after 2nd Iteration

		C1	C2	C3	C4	C5	C6
M^4=	C1	283.131338	340.5673	409.3126	505.564891	744.7667	1064.432
	C2	252.731453	305.7052	366.7913	451.723012	657.4643	942.1743
	C3	191.679143	231.8161	279.474	346.232548	502.6668	714.1062
	C4	174.941816	211.9656	257.3651	322.33722	466.4986	651.6282
	C5	117.534397	141.1787	171.5849	214.706676	317.059	444.7471
	C6	82.02939	98.22449	118.49	146.809531	218.2133	310.8954

Along with the row sums and eigenvector, the difference between the previous eigenvector and the current eigenvector is calculated and is given in Table 4.7.

Table 4.7: Matrix for row sum, eigenvector, and difference for Matrix M^4

	Row Sum		Eigenvector	Difference
	3347.775		0.2564	-0.0024
	2976.59		0.2280	0.0046
	2265.975		0.1736	0.0027
	2084.737		0.1597	0.0031
	1406.811		0.1077	-0.0047
	974.6612		0.0746	-0.0034
Row Total	13056.55	Total	1.0000	

Third Iteration: After squaring the M^4 Matrix, the resulting M^8 Matrix is shown in the following Table 4.8.

Table 4.8: Matrix for M^8 after 3rd Iteration

		C1	C2	C3	C4	C5	C6
M^8=	C1	507986.9	612284.6	739228.6	917838.2	1344779	1906140
	C2	452709.7	545670.2	658791.8	817950.6	1198368	1698649
	C3	344655.8	415431.5	501563.2	622753	912368.8	1293201
	C4	317015.9	382229.7	461493.8	573029.2	839490.9	1189814
	C5	213156.1	256921.1	310203.6	385179	564342.2	799841.6
	C6	147594.7	177895.3	214782.1	266682.8	390747.1	553844.1

The row sums, eigenvectors calculated on M^8 Matrix is given in Table 4.9 along with the difference between the eigenvector of M^4 Matrix and M^8 Matrix.

Table 4.9: Matrix for row sum, eigenvector, and difference for M^8 Matrix

	Row Sum		Eigenvector	Difference
	6028257.30		0.25614312	-0.0003
	5372139.30		0.22826439	0.0003
	4089973.30		0.17378464	0.0002
	3763163.50		0.15989835	0.0002
	2529643.60		0.10748559	-0.0003
	1751546.10		0.07442391	-0.0002
Row Total	23534723.10	**Total**	1.0000	

Now, as there is not much difference between the eigenvector's of the previous two iterations, hence these values are accepted as final values. Now for checking the consistency of the matrix, the next step is to compound the maximum eigenvalue (λ_{max}) which is the mean of λvalues. For consistent matrix λ_{max} >= n. For the proposed maintainability case study as n=6 hence λ_{max} >= 6.

As per Thomas L. Saaty consistency check, as the value of maximum Eigenvalue is 6.51514 > 6 hence the Matrix has been found as consistent.

Step 4: Values of Consistency Index and Consistency Ratio are calculated as given in Equation 4.3 and Equation 4.4, respectively.

$$\text{Consistency Index (CI)} = \frac{(\lambda \text{max} - \text{n})}{\text{n} - 1} = .103028 \quad (4.3)$$

And

$$Consistency\ Ratio\ (CR) = 0.08 < 0.1 \quad (4.4)$$

Hence as per results, the suitability and the usefulness of the proposed maintainability model has been validated.

Step 5: The final computation confirms the proposed maintainability model is consistent and gives the relative ranking of the quality attributes concerning

maintainability in the order of extensibility; reusability; modifiability; analysability; testability and modularity as displayed in Figure 4.3.

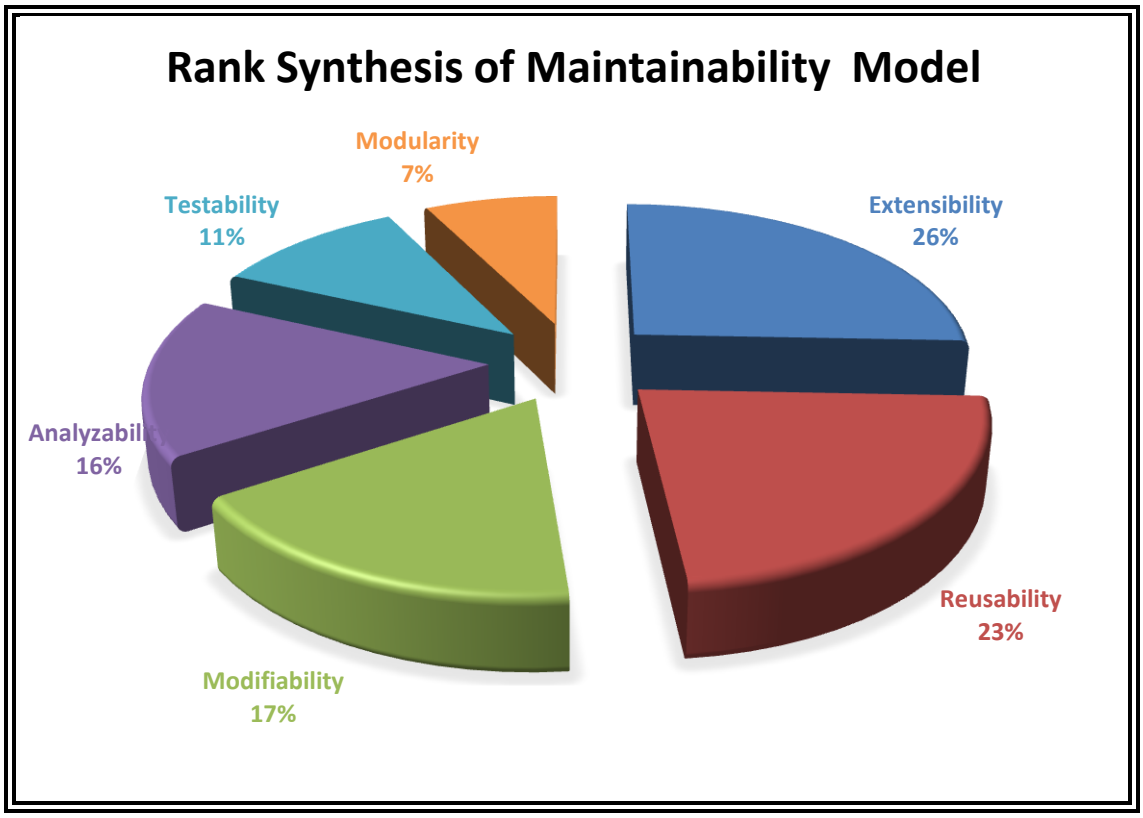


Figure 4.3: Rank Synthesis of Maintainability model

Hence, it establishes that extensibility is a desirable characteristic in the evaluation of maintainability, and validates its inclusion in the maintainability model as a sub-characteristic that needs to be considered.

The validation details of the second proposed characteristic in the proposed model for performance efficiency are specified in the following section.

4.3 OPTIMIZED CODE AS A SUB-CHARACTERISTIC IN PERFORMANCE EFFICIENCY

Performance efficiency can be defined as the performance relative to the number of resources used under the stated conditions. Writing an Optimized code has a positive effect on performance in terms of less response time, increased throughput, reduced memory consumption, and reduced network bandwidth consumptions. A well-

structured program written in a consistent style, free of kludges, developed so that each component is organized and straightforward, and designed makes the product is easy to change and increases the performance of the software.

As well quoted by an early pioneer of software engineering, David Lorge Parnas, who developed the concept of information hiding in modular programming, which is an important element of object-oriented programming today, that

"For much of my life, I have been a software voyeur, peeking furtively at other people's dirty code. Occasionally, I find a real jewel, a well- structured program written in a consistent style, free of kludges, developed so that each component is simple and organized, and designed so that the product is easy to change."

More books have been written about programming (coding) and the principle and concepts that guide it than about any other topic in the software process [2].

Sommerville also identified efficiency as one of the four generalized attributes which are not concerned with, what a program does, but how well the program does it [4].

Coders often due to pressure to meet the deadline of time, try to build the code in a rush, even at the cost to compromise the quality. That type of dirty code might be understandable for a computer, but it is difficult for humans to understand. Due to this messy code, many developers prefer to rewrite the code for any modification or extension rather than performing the difficult task of reading it, understanding it and then comprehending. This scenario explains the essence of optimized coding.

An optimized code is a well-written code that is easy to understand as well as change as it is based on a reader-focused development style. An optimized code has the ability to extend and refactor without much of the effort. To consider any code to be an optimized code, classes, and methods should be small and should focus on an only single functionality. This makes the code to be easily testable with the presence of unit test cases.

In the following section, the proposed performance efficiency model as proposed in Figure 3.2 of Chapter III is validated for the optimized code as an additional

characteristic to evaluate the performance efficiency, which is a vital part for improving the software quality. The validation of the updated performance efficiency model is done using the Analytical Hierarchical Process technique as follows.

4.3.1 Validation

Step 1: Performance Efficiency is decomposed into a hierarchy of sub-characteristics time-behavior as Criteria C_1, optimized code as Criteria C_2, resource utilization as Criteria C_3, and capacity as Criteria C_4, as shown in Figure 4.4. Software build using Java and AspectJ are taken as alternatives.

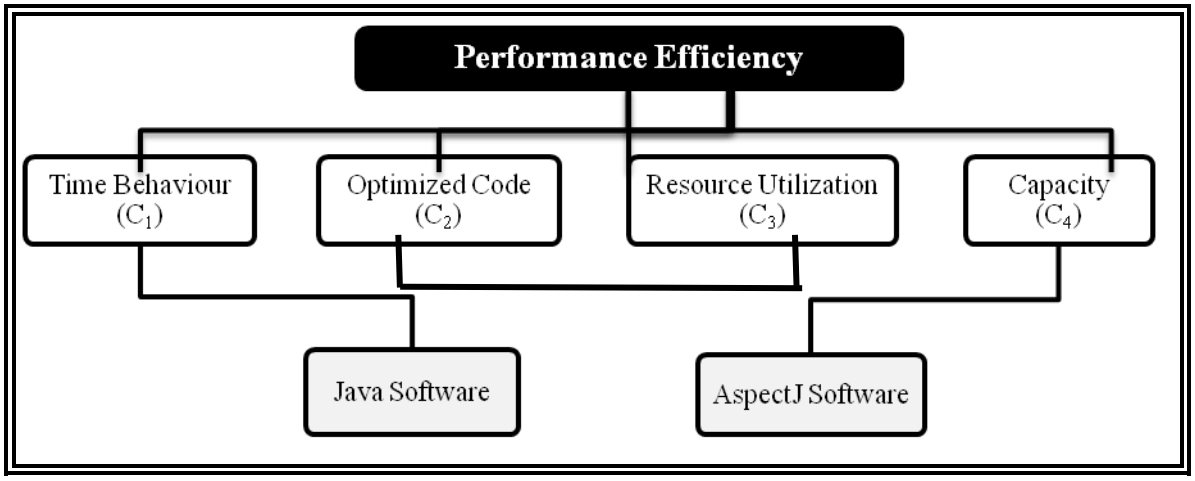


Figure 4.4: Proposed Performance Efficiency model

Step 2: An industrial survey has been conducted using a form (as given in Appendix A Figure A1.6 and Figure A1.7) consisting of 6 comparisons for filling pairwise relative weight of characteristics C1 to C4 in the range of 1 to 9 is provided to each individual. Mean of collected samples of pairwise relative weights is calculated and allocated to sub-characteristics for further evaluation, as shown in Table 4.10.

Table 4.10: Matrix OC for weights allocation to characteristics

		C1	C2	C3	C4
OC=	C1	1	1.595	2.14	1.8858
	C2	0.627	1	3.0867	2.6243
	C3	0.4673	0.324	1.0	2.3575
	C4	0.5303	0.3811	0.4242	1

Step 3: In order to determine the consistency of the allocated weights, eigenvector, and eigenvalue are calculated.

The eigenvector is calculated by squaring the comparison matrix and then calculating the row sum, which is then normalized.

First Iteration: OC^2 Matrix is evaluated by squaring the previous Matrix OC, as demonstrated in Table 4.11.

Table 4.11: Squaring the OC matrix

OC^2=		C1	C2	C3	C4
	C1	4.0000	4.6019	10.0031	13.0024
	C2	4.0879	4.0000	8.6282	13.7077
	C3	2.3878	2.2916	4.0000	6.4464
	C4	1.4977	1.7453	3.1593	4.0000

The row sum and eigenvectors are calculated on the resulting OC^2 Matrix and shown in Table 4.12.

Table 4.12: Row sum matrix and eigenvector of the OC^2 Matrix

	Row Sum		Eigenvector
	31.6074		0.3610
	30.4238		0.3475
	15.1259		0.1727
	10.4024		0.1188
Row Total	87.5594	Total	1.0000

The iteration is repeated until the time difference between the current eigenvector, and the previous eigenvector becomes negligible.

Second Iteration: Squaring the OC^2 Matrix is done, and OC^4 Matrix is evaluated as shown in Table 4.13.

Table 4.13: Squaring the OC^2 Matrix

		C1	C2	C3	C4
OC^4=	C1	78.1713	82.4320	160.8103	231.5851
	C2	73.8355	78.5090	153.2247	218.4349
	C3	38.1252	40.5726	80.0248	114.0316
	C4	26.6601	28.0948	55.3153	79.7643

The row sum matrix, eigenvectors are calculated on the resulting OC^4 Matrix. Also, the difference between the eigenvectors of OC^4 Matrix and OC^2 Matrix is evaluated as shown in Table 4.14.

Table 4.14: Row sum matrix, eigenvector, and difference of the OC^4 Matrix

	Row Sum		Eigenvector	Difference
	552.9987		0.3592	0.0018
	524.0041		0.3404	0.0071
	272.7541		0.1772	-0.0044
	189.835		0.1233	-0.0045
Row Total	1539.5915	Total	1.0000	

Third Iteration : The iteration is repeated until the time, the difference between the eigenvectors of OC^4 and OC^2 Matrix approaches to zero up to three decimal places. Hence the OC^4 Matrix is squared, and the resulting OC^8 Matrix is shown in Table 4.15.

Table 4.15: Squaring the OC^4 Matrix

		C1	C2	C3	C4
OC^8=	C1	24502.1669	25946.3020	50880.3759	72919.0117
	C2	23233.7919	24603.6880	48247.6013	69144.0854
	C3	12067.0475	12778.5585	25059.3028	35912.6987
	C4	8393.8838	8888.5931	17430.8257	24981.0151

The row sum Matrix and eigenvector calculated on the resulting OC^8 Matrix is shown in the following Table 4.16. Also, the difference evaluated for the eigenvector of OC^8

Matrix and the OC^4 Matrix has approached to Zero for three decimal places, as shown in Table 4.16.

Now, after the third iteration, the difference between the current and the previous eigenvector is approaching zero. Hence these values can be accepted as final values.

Table 4.16: Row sum matrix, eigenvector, and difference for the OC^8 Matrix

	Row Sum		Eigenvector	Difference
	174247.8565		0.3593	-0.0001
	165229.1666		0.3407	-0.0003
	85817.6075		0.1769	0.0002
	59694.3177		0.1231	0.0002
Row Total	484988.9482	**Total**	1.0000	

As per Thomas L. Saaty consistency check, as the value of maximum Eigenvalue is 4.2125 > 4; hence, the matrix has been found consistent.

Step 4: Values of Consistency Index and Consistency Ratio are calculated as given in Equation 4.5 and Equation 4.6 respectively.

$$Consistency\ index\ (CI) = \frac{(\lambda \max - n)}{n-1} = 0.070833 \quad (4.5)$$

And

$$Consistency\ Ratio\ (CR) = 0.08 < 0.1 \quad (4.6)$$

Hence as per results, the consistency index and consistency ratio are within the approved limits as given by Thomas Saaty, the suitability and the usefulness of the proposed performance efficiency model has been validated.

Step 5: Results confirm that the chosen quality sub-characteristics are consistent and the relative ranking of the quality attributes for performance efficiency are in the order of time behavior; optimized code; resource utilization and then capacity as presented in Figure 4.5.

Hence, it establishes that optimized code is a wanted characteristic among the software developers and maintainers in the evaluation of performance efficiency, and validates its enclosure in the performance efficiency model as a sub-characteristic that needs to be considered.

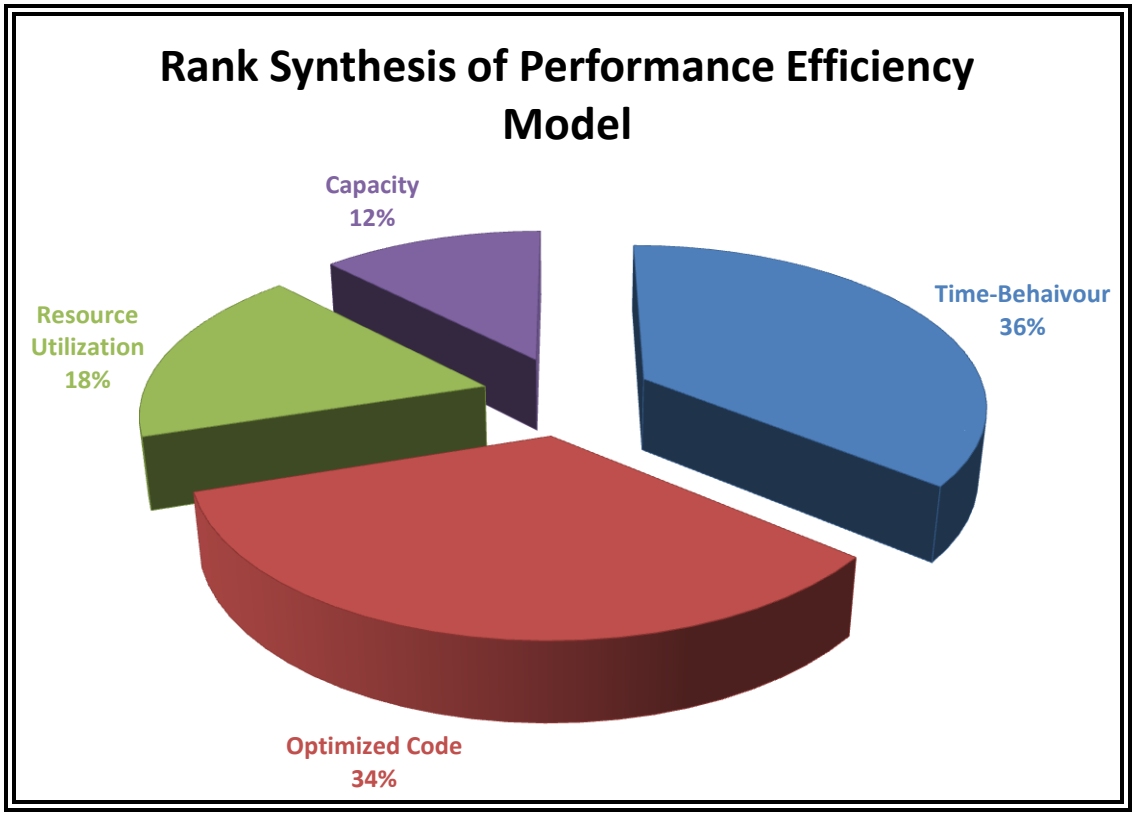


Figure 4.5: Rank Synthesis of Performance Efficiency model

The next section describes the validation details of the proposed software reliability model.

4.4 SCALABILITY AS SUB-CHARACTERISTIC IN SOFTWARE RELIABILITY

Scalability is an essential characteristic of the software quality of a system to be reliable. Scalability is a runtime quality attribute that exhibits the ability of the software system to handle an increase in load by either no adverse impact on the performance of the system or by enlarging the ability. It is not just the ability to operate but to operate efficiently with the satisfactory quality of service even with the

increased load. The scalability of software is quite vital for growing business. It complements and enhances the confidence in the ability of the software system to remain functional over time hence reliability. The key to incorporate scalability in software is the ability of the software to grow along with the increased usage. Scalable software can cater to increased demand for software functions without degrading the performance drastically [72][73][74].

Reliability in ISO 25010 standard quality model for software product was modified from the previous ISO/IEC 9126 standard for the software quality model. The sub-characteristics defined for reliability quality attribute included a new sub characteristic viz availability other than the already existing attributes of maturity, fault tolerance, and recoverability.

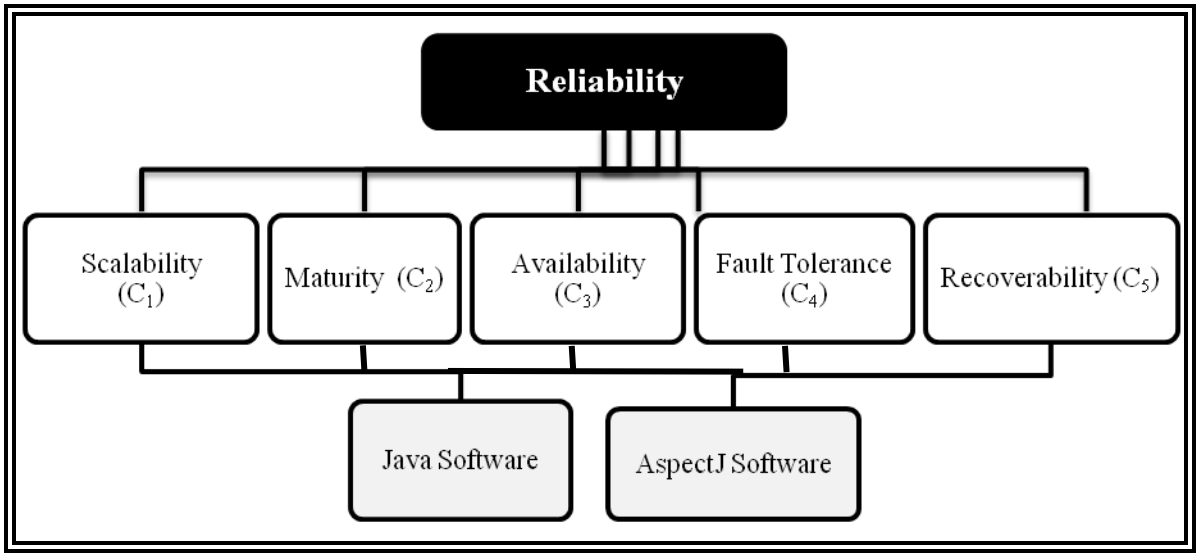


Figure 4.6: Proposed Reliability model

Considering Scalability to be a vital attribute contributing to the reliability of the software to work under adverse conditions, hence a modified reliability model is proposed based on ISO 25010, with the additional characteristic of Scalability as shown in Figure 4.6. The validation of the proposed software reliability model is given in the following sub-section using the Analytical Hierarchical Process technique of the multi-criteria decision making approach.

4.4.1 Validation

Step 1: Software Reliability is decomposed into a hierarchy of sub-characteristics scalability as Criteria C_1, maturity as Criteria C_2, availability as Criteria C_3, fault tolerance as Criteria C_4, and recoverability as Criteria C_5 as modeled in Figure 4.6. Alternatives are considered as software build using two programming languages, namely Java and AspectJ.

Step 2: An industrial survey has been conducted using a form (as given in Appendix A Figure A1.4 and Figure A1.5) consisting of 15 comparisons for filling pairwise relative weight of characteristics C1 to C5 in the range of 1 to 9 is provided to each individual. Mean of collected samples of pairwise relative weights is calculated and allocated to sub-characteristics for further evaluation, as shown in Table 4.17.

Table 4.17: Matrix S for weights allocation to characteristics

		C1	C2	C3	C5	C4
S =	C1	1	2.0946	2.1853	1.4811	0.7687
	C2	0.4774	1	2.201	0.9407	2.1408
	C3	0.4576	0.4543	1	1.4891	0.8947
	C3	0.6752	1.0631	0.6715	1	2.0881
	C4	1.3009	0.4671	1.1177	0.4789	1

Step 3: To determine the consistency of the allocated weights, eigenvector, and eigenvalue are calculated. The Eigenvector for the Matrix S is calculated by squaring the comparison matrix and calculating the row sum, which is then normalized.

First Iteration: The resulting S^2 Matrix after squaring the previous S Matrix is shown in Table 4.18.

Table 4.18: S^2 Matrix after 1st Iteration

		C1	C2	C3	C5	C4
S^2 =	C1	5.0000	7.1158	10.8347	8.5550	11.0696
	C2	5.3821	5.0000	8.4698	6.8913	8.5821
	C3	3.3014	3.8681	5.0000	4.5119	6.2232
	C3	4.8815	4.8208	7.4922	5.0000	7.5718
	C4	3.6596	4.6760	6.4280	4.9884	5.0000

The row sums and eigenvectors evaluated using S^2 Matrix is shown in Table 4.19.

Since Eigen Vector is normalized, hence the sum of the column of the eigenvector is one.

Table 4.19: Row sum and eigenvector of the S^2 Matrix

	Row Sum		Eigenvector
	42.5750		0.2759
	34.3253		0.2224
	22.9046		0.1484
	29.7663		0.1929
	24.7521		0.1604
Row Total	154.3233	**Total**	1.0000

Second Iteration : S^4 Matrix after squaring the previous calculated S^2 Matrix is shown in Table 4.20.

Table 4.20: S^4 Matrix after 2nd Iteration

		C1	C2	C3	C5	C4
S^4 =	**C1**	181.3389	206.0714	303.8672	238.6913	303.9670
	C2	146.8302	169.4119	249.8083	195.9826	250.2874
	C3	98.6320	113.0242	167.3388	131.0627	166.1369
	C3	127.2057	147.3306	217.3147	171.5581	217.7522
	C4	107.3355	121.7140	180.9104	142.4186	183.4147

Table 4.21: Row sum, eigenvector, and difference of the S^4 Matrix

	Row Sum		Eigenvector	Difference
	1233.9359		0.2718	0.0041
	1012.3204		0.2230	-0.0006
	676.1946		0.1490	-0.0005
	881.1614		0.1941	-0.0012
	735.7932		0.1621	-0.0017
Row Total	4539.4055	**Total**	1.0000	

Along with the row sums and eigenvector for S^4 Matrix; the difference between the

eigenvectors of S^4 Matrix and eigenvectors of S^2 Matrix are also evaluated to identify the no change state as given in Table 4.21.

The iterations are repeated until the difference between the current eigenvector, and the previous eigenvector becomes negligible that is up to three decimal places.

Third Iteration : The resulting S^8 Matrix by squaring the previous S^4 Matrix is given in Table 4.22.

Table 4.22: S^8 Matrix after 3rd Iteration

S^8 =		C1	C2	C3	C5	C4
	C1	156101.7048	178787.6594	264292.0213	207736.0800	264909.2736
	C2	127934.7399	146530.0051	216609.5042	170257.4060	217117.6086
	C3	85490.4663	97916.9581	144745.4453	113771.1080	145081.7060
	C3	111329.8402	127514.1089	188498.9952	148163.1408	188941.4535
	C4	92982.3297	106492.6018	157425.5374	123739.2191	157798.7551

The row sums, eigenvectors evaluated are given Table 4.23. Also, the calculated difference between the eigenvector of the S^8 Matrix and the previous S^4 Matrix is approaching to zero for three decimal places, as shown in Table 4.23.

Table 4.23: Row sum, eigenvector, and difference of S^8 Matrix

	Row Sum		Eigenvector	Difference
	1071826.7391		0.2720	-0.0002
	878449.2638		0.2229	0.0001
	587005.6837		0.1490	0.0000
	764447.5386		0.1940	0.0001
	638438.4432		0.1620	0.0001
Row Total	3940167.6684	**Total**	1.0000	

Now, after the third iteration, the difference between the second and third iteration eigenvectors is approaching zero. Therefore, the iterations are stopped here and these values are taken as final values.

As per Thomas L. Saaty consistency check, the input matrix has been found consistent as the value of maximum Eigenvalue is 5.4278 > 5.

Step 4: Values of Consistency Index and Consistency Ratio are calculated as given in Equation 4.7 and Equation 4.8, respectively.

$$Consistency\ Index\ (CI) = \frac{(\lambda max - n)}{n-1} = 0.106951 \quad (4.7)$$

And

$$Consistency\ Ratio\ (CR) = 0.09 < 0.1 \quad (4.8)$$

As per evaluated results for consistency index and consistency ratio are in accordance with the limits given by Thomas Saaty for consistency; therefore, the suitability and the usefulness of the proposed reliability model has been validated.

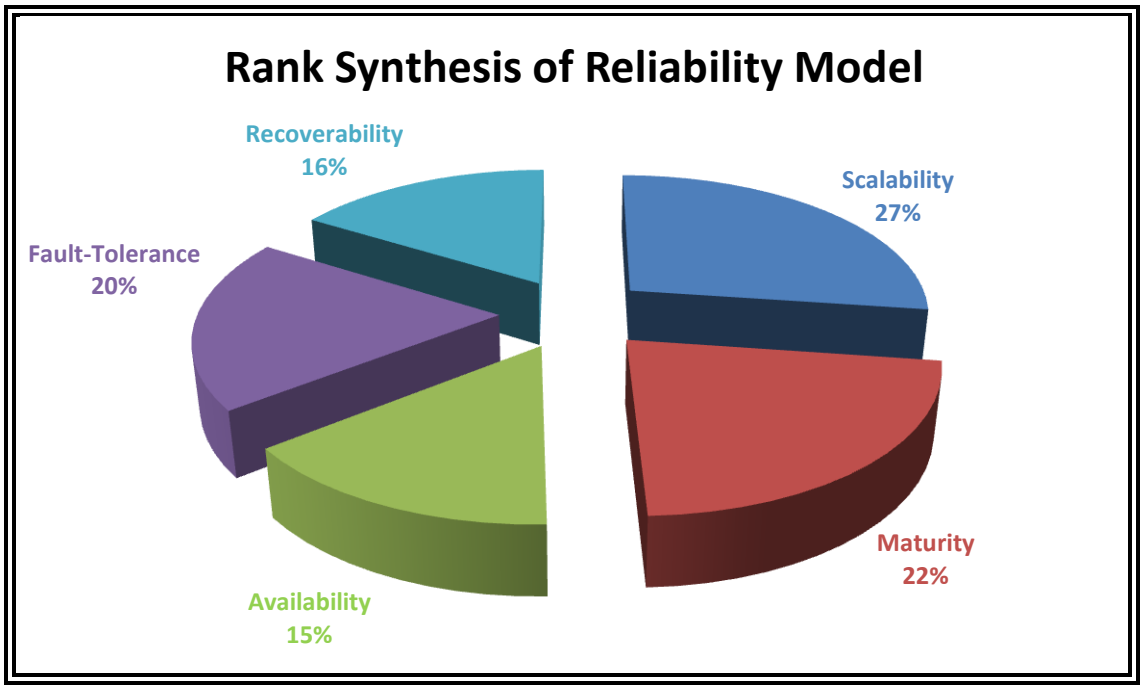


Figure 4.7: Rank Synthesis of Reliability Model

Step 5: Results computed indicate that the chosen quality sub-characteristics for reliability are consistent and the relative ranking of the quality sub-characteristics for proposed reliability model is in the order of scalability, maturity, fault tolerance, recoverability and then availability as depicted in Figure 4.7.

Hence, it ascertains that scalability is a looked-for characteristic among the software industry personnel in the evaluation of software reliability, and so its insertion in the software reliability model as a sub-characteristic is validated and requires to be considered.

The validation details of the proposed usability model are presented in the next section.

4.5 SUPPORTABILITY AS SUB-CHARACTERISTIC IN USABILITY

Users perceive software in a way that it represents its functions. Even minute details in the user interface affect the impression of the overall software product on the customers. One of the fundamental reasons for the aversion among users of the interactive software is it being unfriendly, sophisticated, and low-quality user-interfaces. Software with poor user interface design not only waste the time of its users but also disappoints them and adds to their frustration. Incorporating affable 'ease of use' in software is a tedious task in itself. Ideally, the source code should be written in a manner that reduces the effort required to comprehend its behavior. Most source code programming style guides often strain on enhancing the readability of the software. They focus on following the language-specific conventions to reduce the outlay of source code maintenance [51][52][53].

From a human point of view, source code can be written in a way that affects the effort needed to comprehend its behavior. Many source codes programming style guides, which often stress readability and usually language-specific conventions, are aimed at reducing the outlay of source code maintenance. For a system to be usable, it has to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. That is, the primary notion of usability consists of three sub-goals namely:-

a. Effectiveness, i.e., the capability to produce the desired result,
b. Efficiency, i.e., the extent to which time, effort or cost is well used for the intended task/purpose, and
c. Satisfaction, i.e., more satisfying (pleasant) to use.

Supportability is the capability of the system to provide information helpful for identifying and resolving issues while using the software or when it fails to work correctly. The features for supportability may include user training, customer consulting, upgrades, comprehensive documentation, and online support along with troubleshooting tools, help menus, event logging, and tracing code. Software products that have clear, complete, accurate and consistent documentation of the software product along with the support of help menus and troubleshooting tools and that can be easily traceable; makes the software more effective, efficient and satisfying than before. Hence software products that have sufficient supporting documents make the software more usable and easily acceptable by the customer. Supportability not only increases the chances of the acceptance of the software. Instead, but it is also quite vital at the time of safety [75][76].

Supportability characteristics of earlier software product constituted mainly of maintenance and fixing of errors/issues, but now it encompasses a broad and extensive range of elements including a telephonic or online support system and customer consulting.

Although many sub-characteristics are defined for usability in ISO 25010, one important characteristic is missing, and that is Supportability. Considering the need of the hour, a new usability model is proposed with supportability as a sub-characteristic, in addition to the other sub-characteristics-given in ISO 25010 model, namely appropriateness recognizability, learnability, operability, user-error protection, user interface aesthetics and accessibility as shown in Figure 3.2 of Chapter III. So, supportability is proposed as a sub-characteristic of usability.

The validation of the proposed Usability model is carried out using the technique of multi-criteria decision making approach [19][77], Analytical Hierarchical Process, and is described in the following sub-section.

4.5.1 Validation

Step 1: Software Usability is decomposed into a hierarchy of sub-characteristics appropriateness recognisability as Criteria C_1, supportability as Criteria C_2, learnability as Criteria C_3, operability as Criteria C_4, user error protection as Criteria

C_5, user interface aesthetics as Criteria C_6, and accessibility as Criteria C_7, as given in Figure 4.8. Java build and AspectJ build software are taken as alternatives.

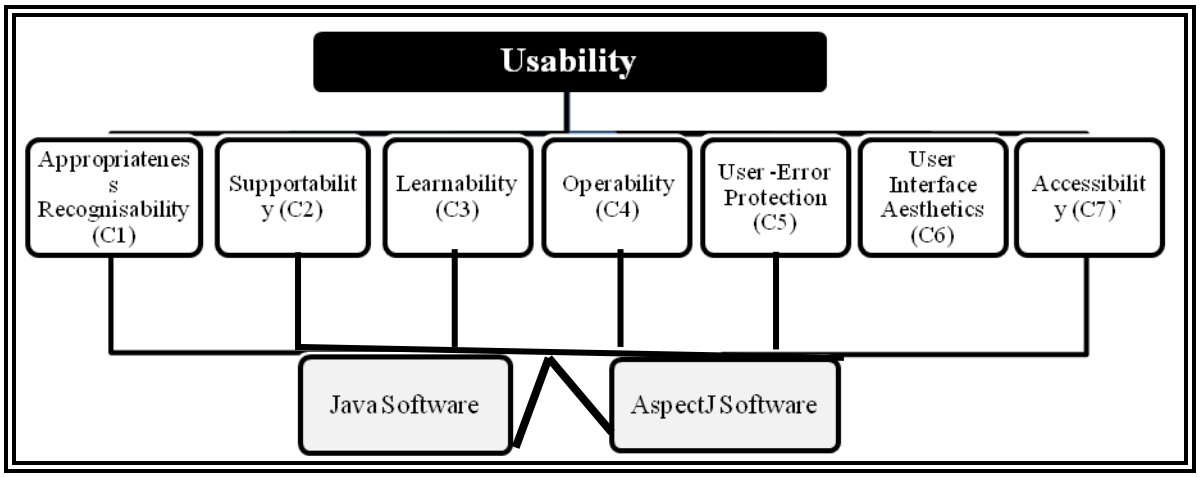


Figure 4.8: New Proposed Usability Model

Step 2: An industrial survey has been conducted using a form (as given in Appendix A Figure A1.8, Figure A1.9 and Figure A1.10) consisting of 21 comparisons for filling pairwise relative weight of characteristics C1 to C7 in the range of 1 to 9 is provided to each individual. Mean of collected samples of pairwise relative weights is calculated and allocated to sub-characteristics for further evaluation, as shown in Table 4.24.

Table 4.24: Matrix SU for weights allocation to characteristics

		C1	C2	C3	C4	C5	C6	C7
SU =	**C1**	1	2.627	2.0301	3.0578	1.5834	3.0518	2.4375
	C2	0.3807	1	1.8051	1.8176	2.0415	3.9207	2.8379
	C3	0.4926	0.554	1	2.9534	2.7639	3.8758	3.3768
	C4	0.327	0.5502	0.3386	1	2.1923	4.2688	3.1161
	C5	0.6315	0.4898	0.3618	0.4561	1	4.2088	4.0954
	C6	0.3277	0.2551	0.258	0.2343	0.2376	1	1.9395
	C7	0.4103	0.3524	0.2961	0.3209	0.2442	0.5156	1

Step 3: In order to determine the consistency of the allocated weights, eigenvector, and eigenvalue are calculated.

Table 4.25: SU^2 Matrix after 1st Iteration

SU^2 =		C1	C2	C3	C4	C5	C6	C7
	C1	7.0000	10.4738	11.9197	19.1054	22.1646	45.2454	41.1172
	C2	5.9832	7.00000	7.5891	12.8908	15.2841	33.8136	34.3276
	C3	6.5829	7.5593	7.0000	11.6723	15.6590	37.4083	37.5661
	C4	5.0921	5.4078	5.1517	7.0000	8.7366	23.8388	26.9919
	C5	4.8363	5.6068	5.3431	7.1025	7.0000	17.7265	21.9263
	C6	1.9019	2.4426	2.3813	3.4269	3.2150	7.0000	7.9758
	C7	1.5286	2.3742	2.3913	3.6436	3.5019	7.2101	7.0000

While synthesis, in iteration, comparison matrix is squared, row sums and eigenvectors are evaluated, and the difference between the current and previous iterations eigenvectors are calculated. The procedure is followed until the time difference becomes negligible. The resulting SU^2 Matrix after squaring the SU Matrix is given in Table 4.25. The row sum and eigenvector evaluated on SU^2 Matrix is shown in Table 4.26.

Table 4.26: Row sum and eigenvector on SU^2 Matrix

	Row Sum			Eigenvector
	157.0261			0.2595
	116.8864			0.1932
	123.4277			0.2040
	82.2188			0.1359
	69.5416			0.1149
	28.3434			0.0468
	27.6497			0.0457
Row Total	605.0958		**Total**	1.0000

Second Matrix: SU^4 Matrix after squaring SU^2 Matrix is displayed in Table 4.27.

Table 4.27 SU^4 Matrix after 2nd Iteration

SU^4 =		C1	C2	C3	C4	C5	C6	C7
	C1	543.2817	672.4641	689.281	1003.9087	113.4044	2578.3007	2745.5044
	C2	389.9157	488.53360	488.2463	732.8692	806.9727	1853.7392	1964.4516
	C3	400.8487	506.0480	507.3060	762.5282	834.0199	1902.9496	2012.0229
	C4	266.3081	339.2825	341.8531	518.2197	569.6632	1289.1941	1348.1184
	C5	239.7195	303.3050	306.2355	487.1135	521.3829	1183.8822	1232.1051
	C6	102.0598	127.6108	128.4498	195.4891	219.6550	502.9137	526.1554
	C7	100.5027	124.2754	124.3680	188.3130	211.6539	488.7754	515.8227

Row sum and eigenvector is evaluated on the SU^4 Matrix is shown in Table 4.28. Also, the difference in the eigenvector of the second iteration and the first iteration is given in Table 4.28.

Table 4.28: Row sum, eigenvector, and difference using SU^4 Matrix

	Row Sum		Eigenvector	Difference
	9326.1449		0.2630	0.0698
	6724.7283		0.1896	-0.0143
	6925.7232		0.1953	0.0594
	4872.6391		0.1318	0.00168
	4253.7236		0.1200	0.0731
	1802.3334		0.0508	0.0051
	1753.7110		0.0495	-0.9505
Row Total	35459.0038	**Total**	1.0000	

Third Iteration: The SU^8 Matrix calculated by squaring the SU^4 Matrix is given in Table 4.29.

The row sum and eigenvector evaluated on the basis of SU^8 Matrix along with the difference between the eigenvector of SU^8 Matrix and the eigenvector of the SU^4 Matrix is given in Table 4.30. Further iterations are not required as the difference is approaching zero up to three decimal places.

Table 4.29: SU^8 Matrix after 3rd Iteration

SU^8 =		C1	C2	C3	C4	C5	C6	C7
	C1	1898965.1	2381074	2388254	3609954	4005579	9171873	9657213
	C2	1373275	1722044	1727262	2610825	2896771	6632643	6983512
	C3	1417869.5	1778056	1783459	2695753	2990850	6847825	7210023
	C4	955633.23	1198485	1202178	1817212	2016148	4615862	4859683
	C5	865289.13	1085163	1088534	1645518	1825829	4180117	4400698
	C6	365616.77	458489.2	459906.1	695233.3	771459.4	1766290	1859528
	C7	355523.72	445797.4	447160.3	675847.5	750077.9	1717441	1808188

As per Thomas L. Saaty consistency check, the input matrix has been found consistent as the value of maximum Eigenvalue is 7.7206 >7.

Table 4.30: Row sum, eigenvector, and difference of SU^8 Matrix

Row Sum	Eigenvector	Difference
33112913	0.2625	-0.0005
23946333	0.1899	0.0002
24723835	0.1960	0.0007
16665201	0.1321	0.0004
15091147	0.1197	-0.0003
6376523	0.0506	-0.0003
8200135.4	0.0492	-0.0003
Row Total 126116088	**Total** 1.0000	

Step 4: Values of Consistency Index and Consistency Ratio are calculated as given in Equation 4.9 and Equation 4.10, respectively.

$$\text{Consistency Index (CI)} = \frac{(\lambda \max - \mathrm{n})}{\mathrm{n} - 1} = 0.120104 \qquad (4.9)$$

And

$$Consistency\ Ratio\ (CR) = 0.09 < 0.1 \qquad (4.10)$$

Evaluated results for consistency demonstrate that the matrix is consistent; hence, the proposed usability model is a valid model.

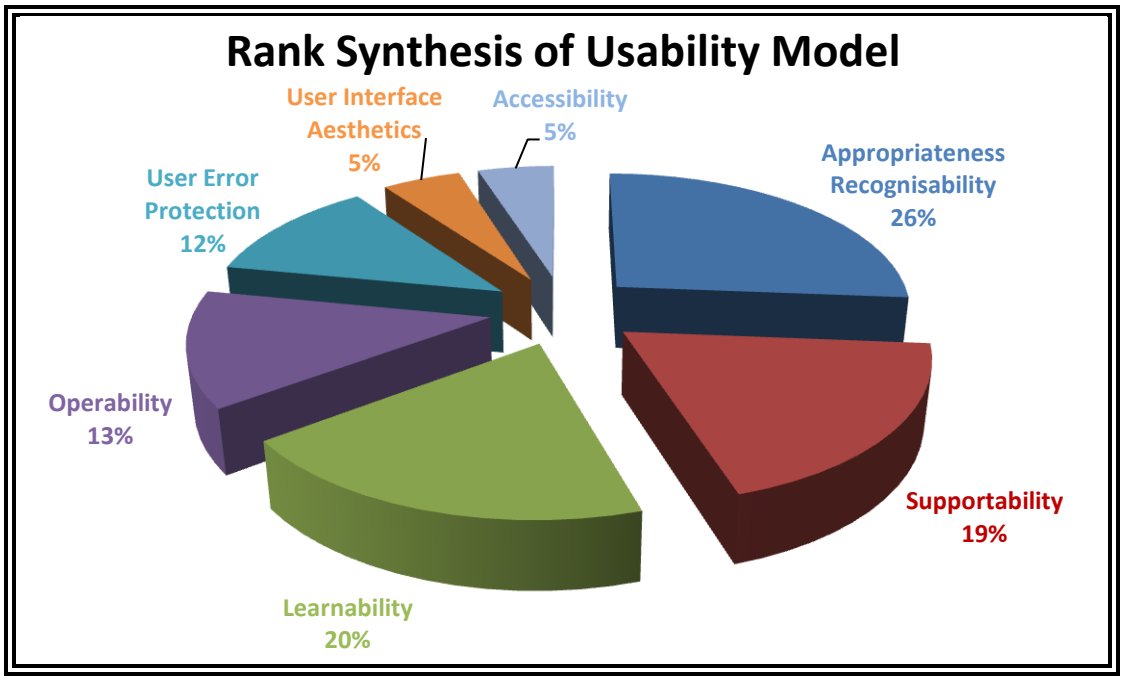


Figure 4.9: Rank Synthesis of Usability Model

Step 5: After the consistency check, alternative factors are evaluated according to their weights, and a final ranking of the factors is done, as shown in Figure 4.9. Evaluated results proved the model to be consistent, and the relative ranking of the factors is appropriateness recognizability, learnability, supportability, operability, user error protection, user interface aesthetics, and accessibility.

Hence, it is apparent that supportability is quite an important characteristic, and its inclusion in the estimation of usability needs to be considered, thereby validating the proposed usability model.

4.6 CHAPTER SUMMARY

In this Chapter, the analytic hierarchy process (AHP) technique is applied to validate and evaluate the relevance of the proposed four characteristics under the higher-level characteristics of the proposed Quality model for Aspect-oriented approach. Four hierarchical models as a base model are designed for the application of AHP. To conduct the AHP technique, the surveys on nearly 110 participants from the IT

industry have been carried out, and the value of pairwise relative weights for the characteristics is taken. The mean of the collected samples has been considered as pairwise relative weights.

The case study validates the suitability and the usefulness of all the four proposed characteristics of the proposed model. The final relative ranking of quality attributes approves the inclusion of the proposed characteristics as desirable characteristics in the modified quality model.

In the following Chapter V, the existing object-oriented software metrics suite is used to determine the maintainability and reusability of software systems developed using AOP approach. For this purpose, the statistics for software, Spacewar, developed in Java and AspectJ are collected, and the comparison of the software quality characteristics reusability, complexity, and maintainability is made.

CHAPTER V

INVESTIGATION OF REUSABILITY AND COMPLEXITY OF AOP SYSTEMS

5.1 INTRODUCTION

An important reason to write good software is to provide quality service to the customers. Kitchenham [78][79] (1989 a, b) refers to software quality as "fitness for needs" and that it is "hard to define, impossible to measure, easy to recognize." However, measuring software quality is the prime difficulty faced by IT professionals. Measurements play a vital role in understanding, controlling, and improving the performance of software systems. As computers grow powerful, software also tends to become more sophisticated and powerful. Measurement of quality of such systems is a big challenge. Maintainability and reusability are essential attributes of quality. Metrics have been used to assess the quality of software systems. Software metrics provide a qualitative measure of the degree to which a system, component, or process possesses a given attribute [1][2][71].

Nowadays, reuse has been playing a critical role in the design and development of software systems. An important question faced by the software engineering community while developing and maintaining software is where the complexity and reusability are more? Selection of programming language and tool to build software plays an important role not only in the coding process but also on the reusability of the resulting software product. Java has been used extensively to develop object-oriented software, whereas AspectJ has been deployed to write Aspect-oriented systems. However, the issue which approaches lead to better maintainable and reusable systems has not been investigated [80][81].

The next section describes the various software metrics used for the comparison purpose of the software programmed in an object-oriented language and aspect-oriented language.

5.2 SOFTWARE METRICS USED

With the usage of metrics, the quality of the software product can be assessed on a continuous basis and leads to informed decision making. Out of various available metrics, selection of the appropriate metrics ensures the development of software with sustainable quality sustaining customer satisfaction, and improved business in the market [82]. The metrics for object-oriented software systems conform to the characteristics of object-oriented software like encapsulation, inheritance, polymorphism, information hiding, massaging, localization, and object abstraction techniques.

CK metric suite is for measuring the complexity about its impact on efficiency, reusability, maintenance, etc. of an Object-Oriented System as they have underline methods to define class coupling and class cohesion. Various metrics defined are WMC (Weighted Methods per Class), RFC (Response For a Class), LCOM (Lack of Cohesion of Methods), CBO (Coupling between object classes), DIT (Depth of Inheritance Tree), NOC (Number of Children) [35].

R.Martin metric suite is to determine the interdependency between the subsystems of the object-oriented system design. Highly interdependent system designs are less reusable and challenging to maintain due to their rigidity. Various metrics defined are instability, afferent coupling, efferent coupling, abstractness, and normalized distance [83].

Lorenz and Kidd metrics suite is to identify various characteristics within a class highlighting aspects of the class abstractions. These metrics further help in determining where corrective action may be taken. Metrics defined are characterized into size oriented, inheritance-based, internal to the class, and external to the class. Various metrics defined are class size, number of operations overridden by the subclass, number of operations added by the subclass and specialization index [36].

External Quality characteristic of a software system can be ascertained by combining the metrics of its sub-characteristics. Depending on the programming methodology used, there are numerous metrics available to measure the quality attributes. As Aspect-Oriented Programming is developed on the basis of Object-Oriented

Programming hence, the metrics applicable on Object-Oriented Software System are also relevant for Aspect-Oriented Software Systems and therefore can be used for the external quality measurement in a certain way.

The comparative analysis of the various metrics is based on the exemplary software coded in both Java and AspectJ. The calculations of the metrics have been carried out using the Metrics 1.3.6 tool [122] at the Eclipse Platform (as presented in Appendix B Figure A 2.1, Figure A 2.2, Figure A 2.3 and Figure A 2.4). The statistical comparison of the metrics collected and visualized is presented in the next section.

5.3 COMPARATIVE ANALYSIS OF VARIOUS METRICS

Multiple object-oriented metrics proposed by authors have been determined to estimate the impact of aspect orientation on the software development methodology. Until date, there has been relatively less work done in the area of providing the means of comparison between Aspect-oriented systems and their Object-Oriented equivalents. In order to support such comparison, Spacewar software developed using aspect-oriented software, AspectJ, and Spacewar software developed using object-oriented software, Java is used for the comparison purpose. The UML diagrams for Spacewar Java and AspectJ are given in Figures 5.1 and 5.2, respectively.

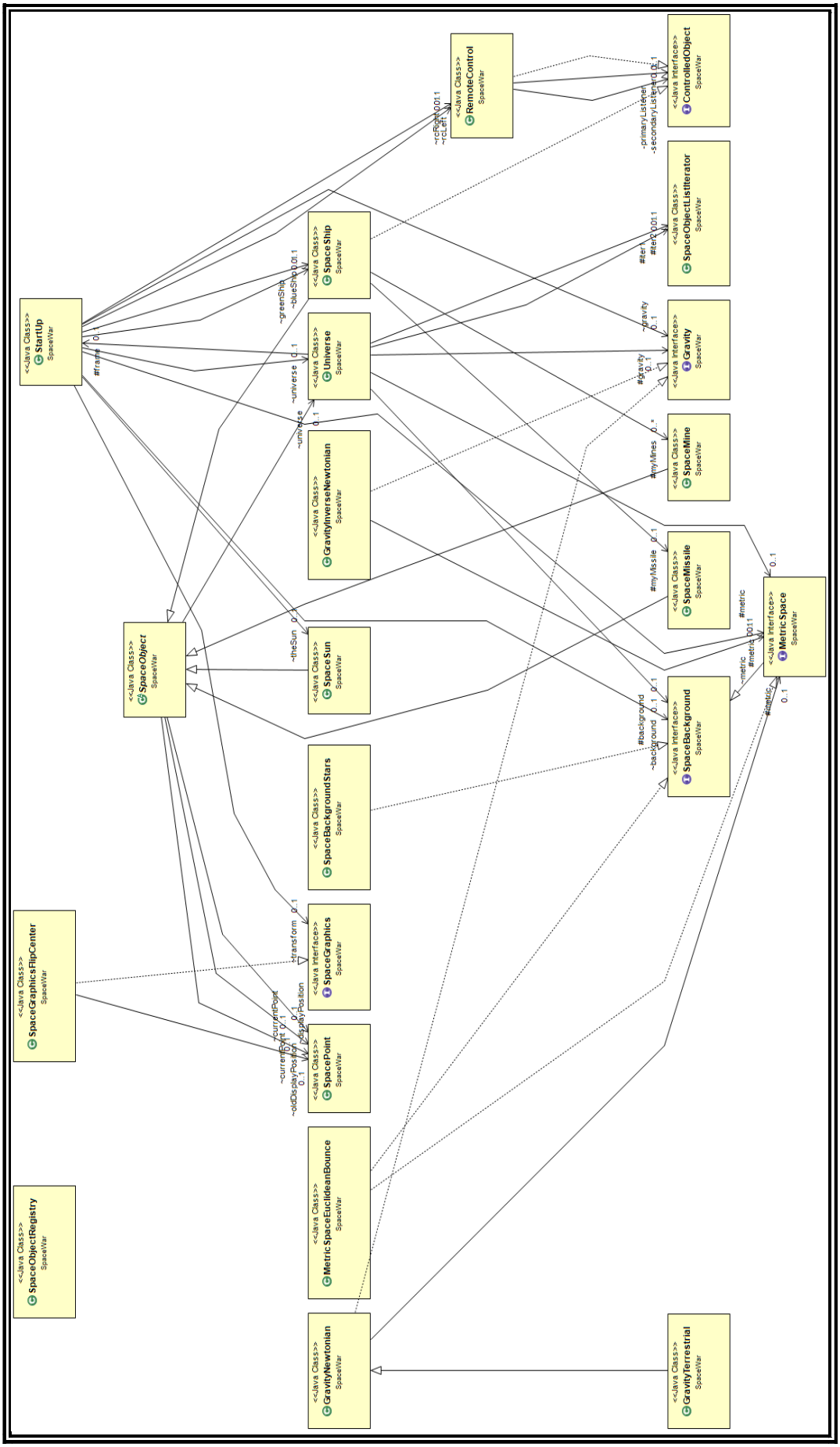


Figure 5.1: UML for Spacewar Java

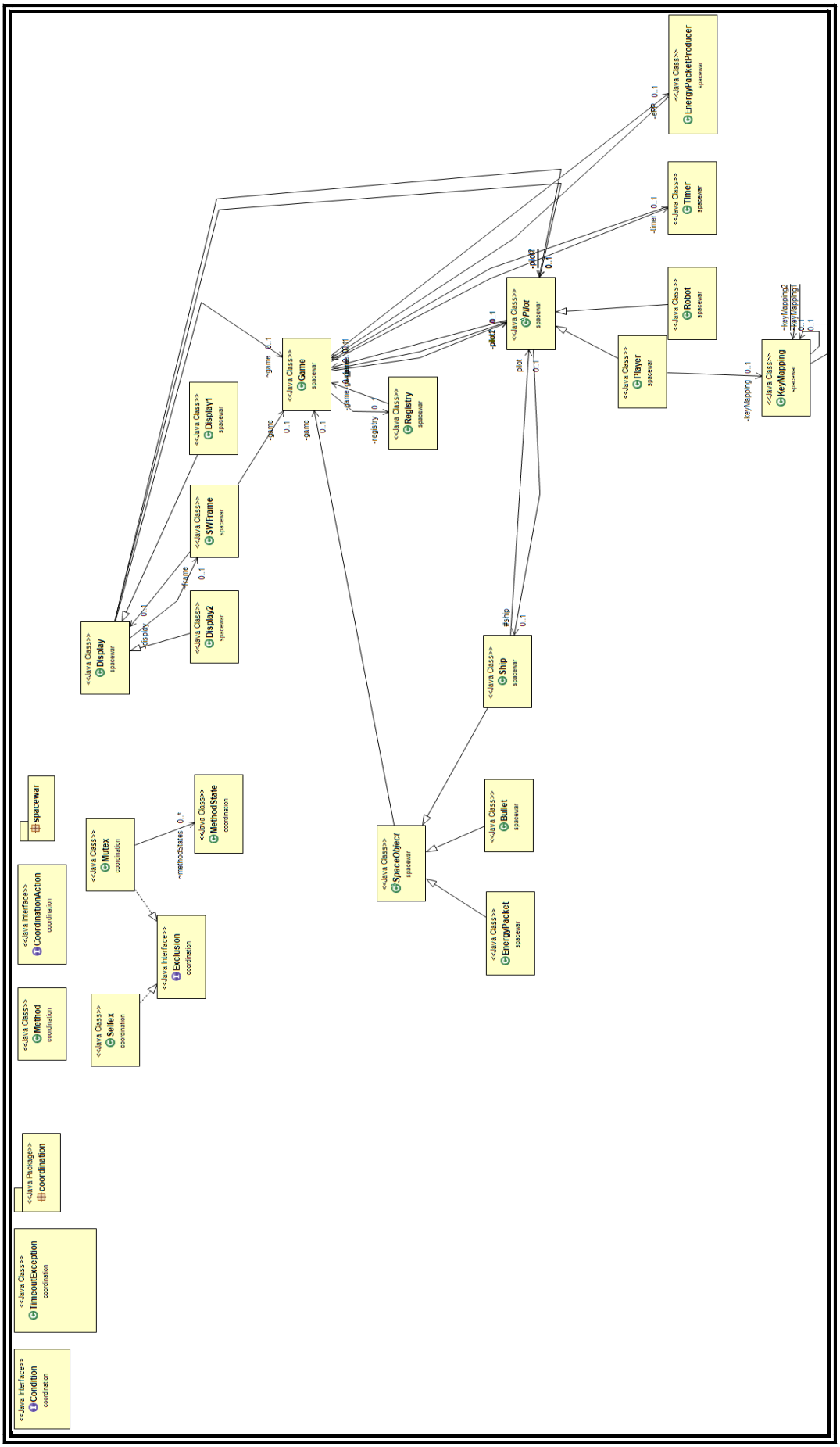


Figure 5.2: UML for Spacewar AspectJ

The mean of the metrics for both the software version, Spacewar software implemented in Java and AspectJ language are collected and are shown in Table 5.1.

Table 5.1: Mean of the Statistics collected for Spacewar (Java) and Spacewar (AspectJ)

Metrics	Spacewar Java	Spacewar AspectJ
Number of Parameters	1.043	0.851
Number of Static Attributes	1.353	1.483
Efferent Coupling	1	1
Specialization Index	0.145	0.191
Number of Attributes	5.824	2.103
Abstractness	0.227	0.222
Normalized Distance	0.227	0.278
Number of Static Methods	0.647	0.207
Depth of Inheritance Tree	1.941	1.828
Instability	1	0.5
McCabe Cyclomatic Complexity	1.851	1.934
Nested Block Depth	1.394	1.331
Lack of Cohesion of Methods	0.352	0.296
Method Lines of Code	6.798	4.961
Number of Overridden Methods	0.235	0.31
Afferent Coupling	0	1
Number of Children	0.294	0.31

Graphical representation of the comparison of both the software (aspect and java) for mean values are shown in Figure 5.3.

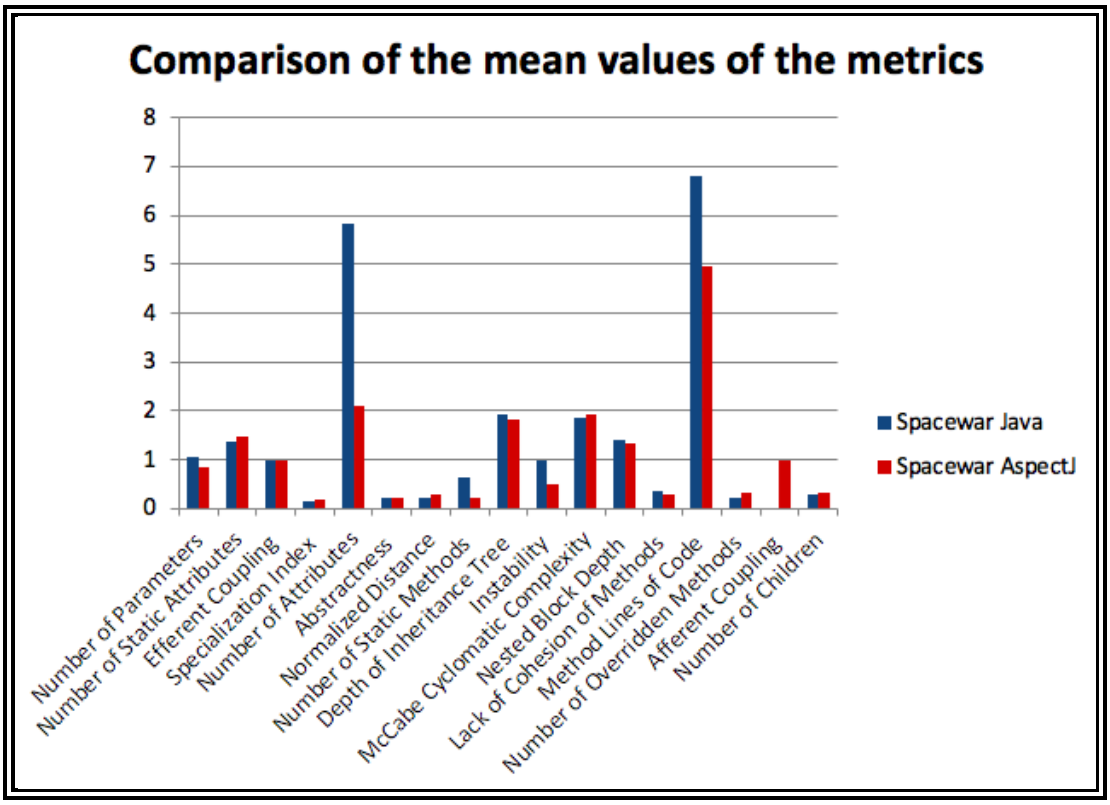


Figure 5.3: Displays the mean values of the collected metrics for Spacewar (AspectJ and Java)

The metrics gathered on the whole package level are listed in Table 5.2.

Table 5.2: Package-level Statistics collected for Spacewar (Java) and Spacewar (AspectJ)

Metrics	Spacewar Java	Spacewar AspectJ
Number of Classes	17	29
Number of Interfaces	5	3
Number of Packages	1	2
Number of Children	5	9

The graphical representation of the package level metrics collected is displayed in Figure 5.4.

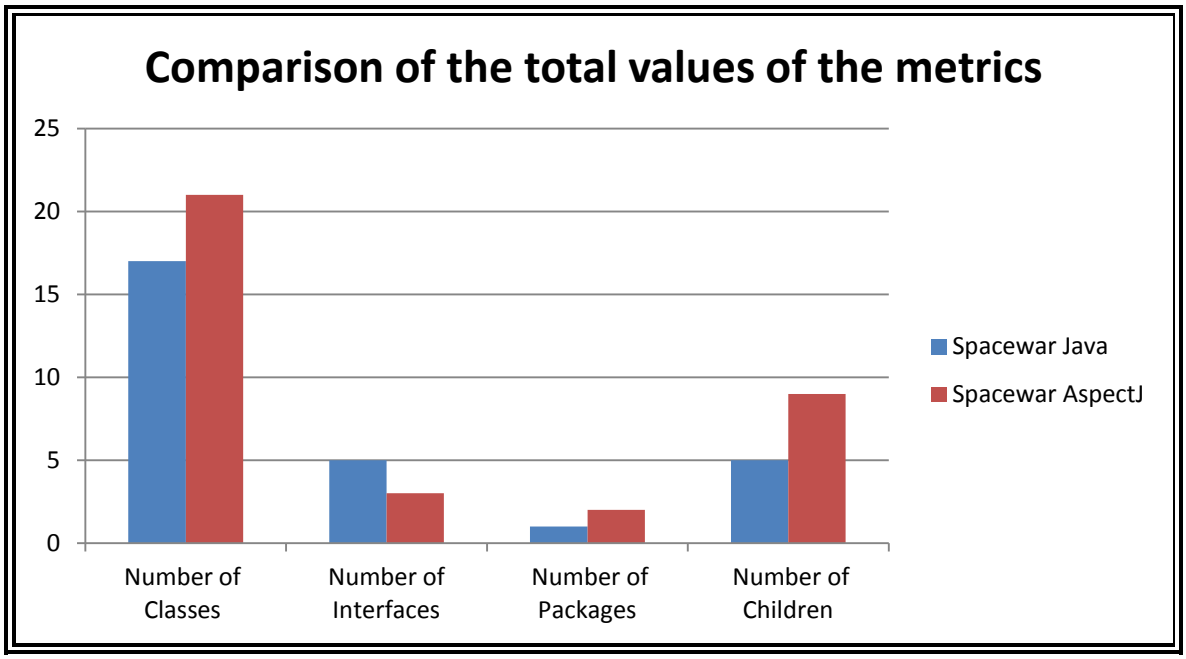


Figure 5.4: Displays the package level metrics collected for Spacewar (AspectJ and Java)

The list of total values of the metrics collected for Spacewar implemented in Java and AspectJ concerning methods is given in Table 5.3. Also, the graphical representation of the same is given in the following Figure 5.5.

Table 5.3: List metrics with respect to methods

Metrics	Spacewar Java	Spacewar AspectJ
Total Lines of Code	1991	1427
Method Lines of Code	1278	898
Weighted methods per Class	348	350
Number of Methods	177	178
Number of Static Methods	11	6
Number of Overridden Methods	4	9

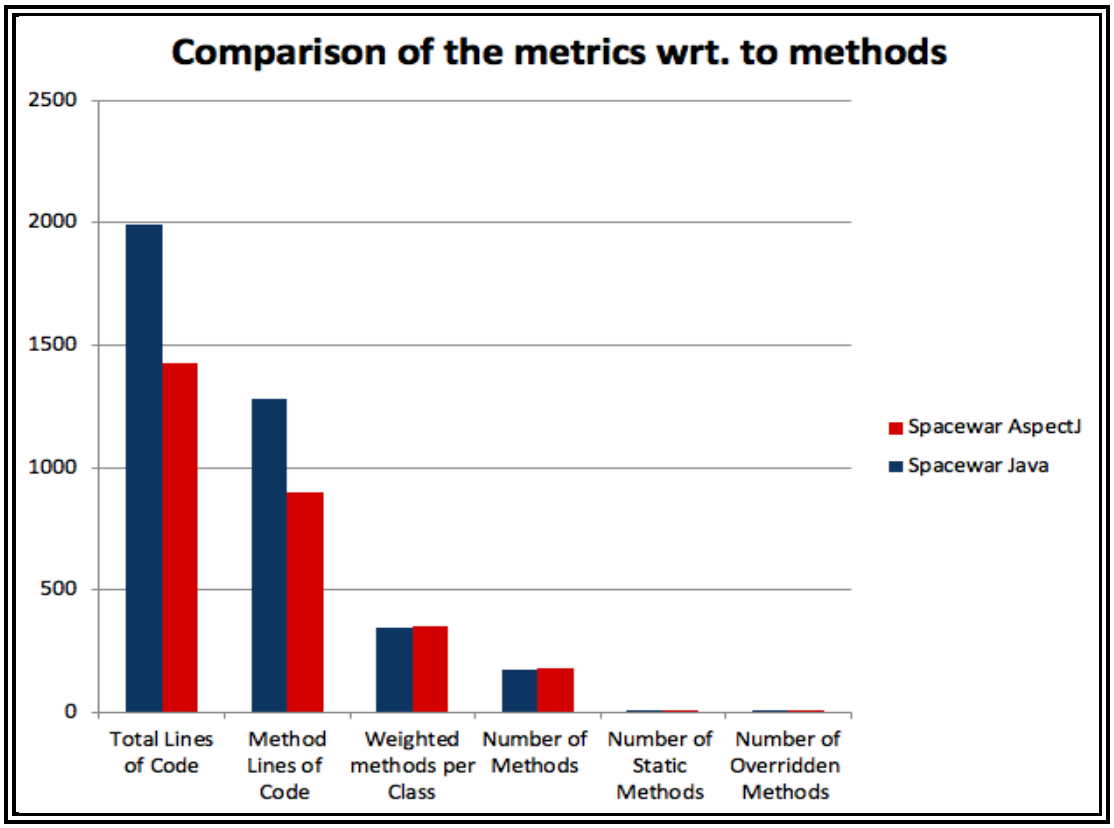


Figure 5.5: Displays the metrics collected with respect to methods

The metrics collected for the exemplary software for both Java version and AspectJ version, concerning attributes used in the respective software versions are collected and given in the following Table 5.4.

Table 5.4: Metrics collected with respect to attributes

Metrics	Spacewar Java	Spacewar AspectJ
Number of Static Attributes	23	43
Number of Attributes	99	61

The graphical representation of the metrics collected concerning attributes used in Spacewar Java version and AspectJ version is given in Figure 5.6.

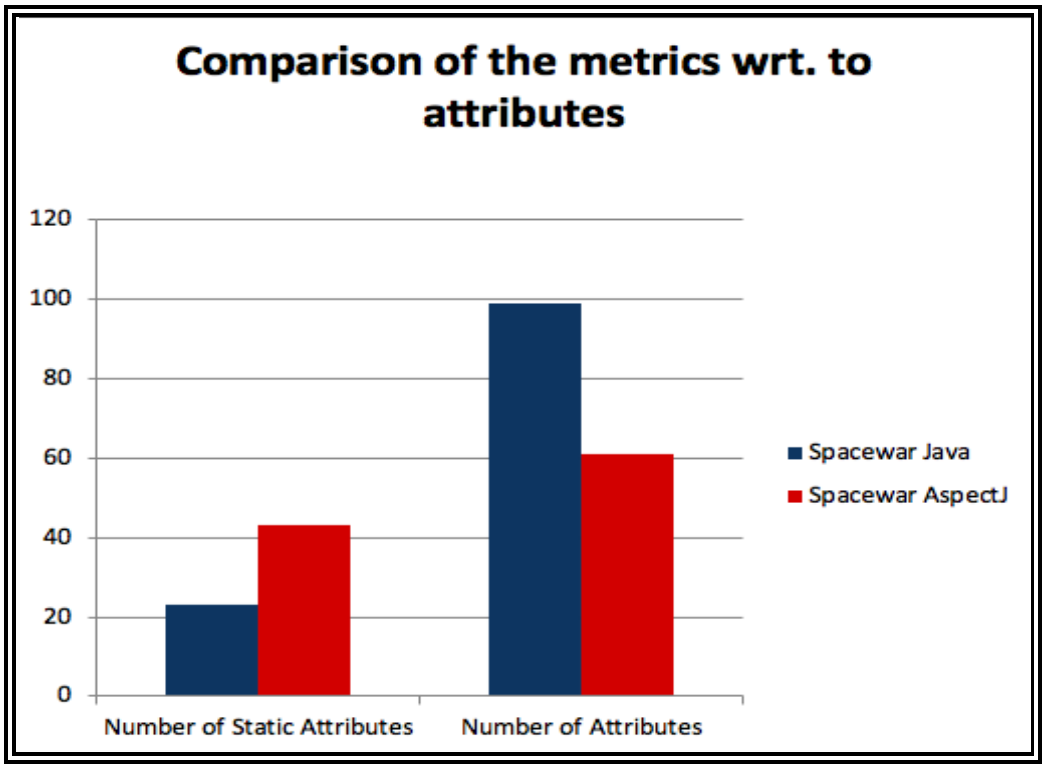


Figure 5.6: Displays the metrics collected with respect to attributes

The discussion and analysis of the collected metrics and inferences concluded out of those measurements with reference to the given reference range as listed in Table 5.5 and is presented in the following section.

5.4 METRIC ANALYSIS

The preferable ranges of the metrics collected are highlighted in the following Table 5.5.

Table 5.5: Preference range of the metrics

Metrics	Preferable
Number of Parameters	Low
Number of Static Attributes	Low
Efferent Coupling	Low
Specialization Index	0<=SIX<=12
Number of Classes	Relative
Number of Attributes	Low
Abstractness	0<A<0.5
Normalized Distance	Low
Number of Static Methods	Low
Number of Interfaces	Limited
Total Lines of Code	Relative
Weighted Methods per Class	Low
Number of Methods	High
Depth of Inheritance Tree	<6
Number of Packages	Relative
Instability	0<I<1
McCabe Cyclomatic Complexity	Low
Nested Block Depth	<=5
Lack of Cohesion of Methods	Low
Method Lines of Code	Relative
Number of Overridden Methods	Low
Afferent Coupling	Low
Number of Children	Relative

Out of the various metrics collected, the results obtained for both AspectJ software and Java software shows the positive inclination towards AspectJ software in Table 5.6.

Table 5.6: Comparative Metric Analysis

ATTRIBUTE	SPACEWAR ASPECTJ	SPACEWAR JAVA
Stability	√	
Complexity		√
Reusability	√	
Maintainability	√	
Cohesion	√	
Coupling		√

5.4.1 Stability

Instability metric indicates the package resilience to change in the software. The range of the metric is 0 to 1, where 1 is for the completely unstable package. Instability metrics for Spacewar Java program is double the value for Spacewar AspectJ. Hence, Spacewar AspectJ software is more stable.

5.4.2 Reusability

Reusability can be analyzed by studying the number of classes, interfaces, and parameters used in the software along with the number of children and the depth of inheritance tree metric for the software.

The total number of classes and interfaces used in a package indicates the extensibility of the software. The value for the total number of classes and interfaces used in the AspectJ Spacewar software is significantly higher than that of Spacewar Java software.

Also, numbers of parameters used in AspectJ Spacewar software are in controlled number as compared to the Java software.

Number of Children (NOC) and Depth of Inheritance Tree (DIT) are used in depicting the use of inheritance in the software. Under controlled values, Inheritance promotes reusability. NOC depicts the width of the inheritance, while DIT depicts the depth of the inheritance. NOC value for Spacewar AspectJ software comes out to be .310

(mean) while for Spacewar Java software it is .294 (mean). Also, DIT is 5 (max) for the AspectJ version as compared to 6 (max) for the Java version. The upper limit for DIT is <6 generally.

Hence these values show that the Spacewar AspectJ software is more reusable and extensible as compared to the Java one.

5.4.3 Maintainability

Maintainability of the software can be analyzed by evaluating the Cyclomatic Complexity, DIT, Class coupling, LOC, and cohesion metrics for the software.

Cyclomatic complexity should be low in the range of 0 to 10. It is more for AspectJ software than Java software.

DIT should be low in the range of <6. It is acceptable for the Spacewar AspectJ as the value is 5(max).

Low-Class Coupling is better, and an attempt should be made to minimize it. The class coupling can be measured in two ways: one Efferent coupling and another Afferent coupling. Efferent coupling is the number of classes in the package that is dependent on the classes outside the package. This value is the same (1 max) for both the versions. Afferent Coupling is the number of classes outside the package that is dependent on the classes inside the package. Here the value of coupling for AspectJ Spacewar software is more as the AspectJ language involves the weaving of aspects into classes. It can, therefore, be concluded that aspectization improves the cohesion of software but introduces coupling between base and aspectual units of encapsulation.

Lines of code (LOC) metric for AspectJ software is significantly lesser than the Java software as a repetition of code is avoided due to the usage of aspects.

Value for Cohesion metric for the modules should be more to localize the effect of change in the software and to increase the modularity of the software. Hence the value for Lack of cohesion of Methods (LCOM) metric should be low. LCOM value for AspectJ software is lesser than that for Java software, which indicates the better

cohesion and modularity of the software, reducing the ripple effect of change in the software.

Considering all these factors, it is assured that the Spacewar software created using AspectJ software better modular, stable, easily maintainable, and modifiable; however, at the cost of complexity caused due to higher coupling.

Three novel metrics in order to fill the research gap of the metrics for the proposed characteristics of the novel quality model are presented in the following Chapter VI concerning, Aspect-oriented System Coupling, Supportability, and Extensibility.

CHAPTER VI

NOVEL METRICS FOR AOP

6.1 INTRODUCTION

All the software quality attributes require their corresponding software metrics to analyze and evaluate the extent of the presence of that particular quality attribute in the software product. Further, these quantitative measures may help the programmers and coders to identify as well as rectify the weakness of the software product. With an aim to fill in the research gap, three metrics are proposed in the following sections concerning, Aspect-oriented System Coupling, Supportability, and Extensibility.

In the subsequent section, the coupling metric for a complete aspect-oriented software system is presented in detail.

6.2 AO COUPLING METRIC

Any software system is composed of entities and the relationships between those entities. The relationship can be inbound (in-between attributes of an entity) or outbound (in-between two entities). The degree of relationship in-between a single entity is called cohesion. A strong inbound relationship depicts how much an entity stands alone. However, as different entities of a software system communicate with each other; coupling comes into the picture. It describes the relationship between two entities in a software system. Coupling is an essential feature of any system, and it cannot be eradicated from the software system without affecting the performance, readability, or maintenance of the software. However, the high coupling should be avoided, as it results in heavy dependency of an entity on another entity. Loose coupling between entities is preferred, as it displays more independence among entities and reduces the chance and scope of a ripple effect caused by any change in an entity.

In procedural programming systems, the degree of coupling is calculated based on the extent to which a module/subroutine has access to another module/subroutine. In

object-oriented programming, the concept of encapsulation was introduced to control coupling. Encapsulation proved to be an essential technique for achieving loose coupling. Classes hide the internal details from other classes, and strict interfaces were used for communication with other classes. The Coupling Between Objects (CBO) metric is used to depict a number of classes referred to or used by a particular class. A high CBO value for class A shows that class A is highly dependent on other classes and that any change in that class is likely to affect various dependent classes, either directly or indirectly. Hence, it makes it difficult to maintain and expand the system [35].

The basis of encapsulation in object-oriented programming is to encapsulate the identified entities into classes. In this framework, the implementation of crosscutting concerns tends to increase the system interdependencies, which consecutively increase the coupling. In Aspect-oriented programming, the encapsulation is not only composed of classes, but cross-cutting concerns are also identified and encapsulated as aspects. This approach allows for the apparent isolation and reuse of code that implements cross-cutting concerns with the usage of aspects [97][98].

A brief revisit of the work done by other researchers for defining metrics for Aspect-oriented software systems, especially coupling, is given in the following sub-section.

6.2.1 Existing Aspect-Oriented Coupling Metric

Significant contributions for analyzing and proposing coupling metrics for Aspect-oriented systems are summarized in Table 6.1.

- Zhao [37][38] used the aspect dependency graph framework to identify interdependencies between aspect and classes and to define measurements for aspect coupling. However, the defined coupling metric did not consider the dependency between aspects or classes.

- Ceccato and Tonella [39] extended the object-oriented CK metrics suite for aspect-oriented systems. Classes and aspects were used as a module, whereas methods, advice, and introductions were used as operations. Various coupling metrics for AOP were inspired from the CK metric suite, such as CMC (Coupling on Method Calls) and CFA (Coupling on Field Access). A separate metrics for measuring the

coupling of aspect as well as introduction were also defined, namely, CAE (Coupling on Advice Execution) and CIM (Coupling on Intercepted Module).

- Bartsch and Harrison [40] evaluated the five Aspect-oriented coupling metrics proposed by Ceccato and Tonella [39] and claimed that only CDA is completely valid from the perspective of measurement theory.

- Kumar et al. [47] proposed the coupling metrics for generic AOS based on the connections between the elements. The proposed metrics were CoAT (Coupling on Attribute Type), CoPT (Coupling on Parameter Type), CoAR (Coupling on Attribute Reference), CoOI (Coupling on Operation Invocation), CoI (Coupling on Inheritance) and CoHA (Coupling on High-Level Association).

Table 6.1: Summary of AOP metrics and features

Author	Zhao and Xu [37][38]	Ceccato and Tonella [39]	Sant' Anna et al. [43][44]	Kumar et al. [47][48][49]
Coupling Metric	**YES**	**YES**	**YES**	**YES**

A summary of software measures proposed by various authors with respect to features is given in Table 6.1. The primary focus of the current coupling metrics defined is based on fields, methods, classes, or aspect as a standalone entity. Only a few metrics exist for the measurement of software quality attributes at a higher level of abstraction in AOSs. Hence, one metric for complete Aspect-oriented Software System Coupling is proposed below.

6.2.2 Proposed Aspect-Oriented System Coupling Metric

Software measurement assessment plays an integral part in first understanding and then controlling the software development process and product in order to improve them. Software metrics are considered to be the critical indicator of the absence/presence of software attributes in the end software product, as well as the extent of their presence. One of the reasons for non-acceptance of the software

measure by the academic or industry is a lack of clear definitions of the concepts that are used in defining software measures [92][93][99][100].

A theoretical framework for defining the Aspect-oriented Coupling Metric is provided in the following subsection. It gives the proposed definitions for the elements and relationships associated with Aspect-oriented software that will support in building the coupling assessment framework.

6.2.2.1 Theoretical Framework

The first crucial activity in proposing software metric is to give clear, unambiguous, and precise definitions of relevant concepts, which acts as a vital foundation for laying down an assessment framework.

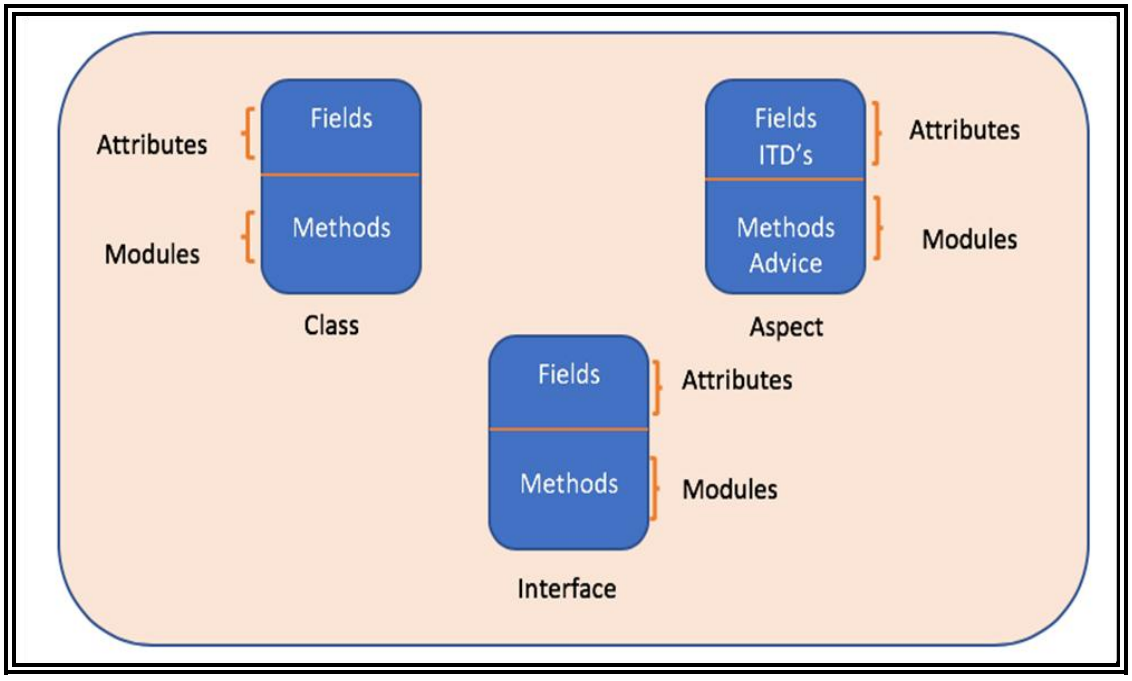


Figure 6.1: Elements of AOS

Defining a meaningful, rational, and sound software measure is essential. Some basic definitions related to the aspect-oriented software system that is used for defining the aspect-oriented system coupling metric and for validating against Briand [101] properties are proposed below:

- *System*

Definition 1: An Aspect-Oriented System S consists of a set of elements E and relations R between the elements as given in Equation 6.1.

$$S = \langle E | R \rangle \quad (6.1)$$

- *Element*

Definition 2: The elements E of an Aspect-Oriented System consist of classes, aspects, and interfaces, as shown in Figure 6.1.

- *Relation*

Definition 3: Relation R(i , j) is equal to one if element i and element j are related to one another. The different types of relations can be:

Class – Class	*Aspect – Class*
Class – Aspect	*Aspect – Aspect*
Class – Interface	*Aspect – Interface*

- *Module*

Definition 4: For each element e ϵ E, let M(e) be the set of modules in e that may consist of –

Methods in Class e or Methods in Interface e or Advice in Aspect e as presented in Equation 6.2.

$$M(S) = \bigcup_{e \in E} M(e) \quad (6.2)$$

- *Attribute*

Definition 5: For each element e $\in$ E, let A(e) be the set of attributes of element e as given in Equation 6.3

$$A(S) = \bigcup_{e \in E} A(e) \quad (6.3)$$

That may consist of –

Attributes in class e or

Attributes in interface e or

Intertype declarations in aspect e

Now, after laying the foundation of the basic definitions to be used for metric proposal, Aspect-oriented metric for measuring the coupling of the system is hereby proposed.

6.2.2.2 New Proposed AO Coupling Metric

The proposed Aspect-oriented System Coupling CO_{AO} metric is defined as shown in Equation 6.4.

$$CO_{AO} = (CO(A) + CO(M)) / n(n-1)) \quad (6.4)$$

Where CO(A) represents Attribute coupling, CO(M) represents Module Coupling, and n represents the total number of elements.

The various steps to measure and analyze the proposed aspect-oriented system coupling metric CO_{AO} are illustrated in Figure 6.2.

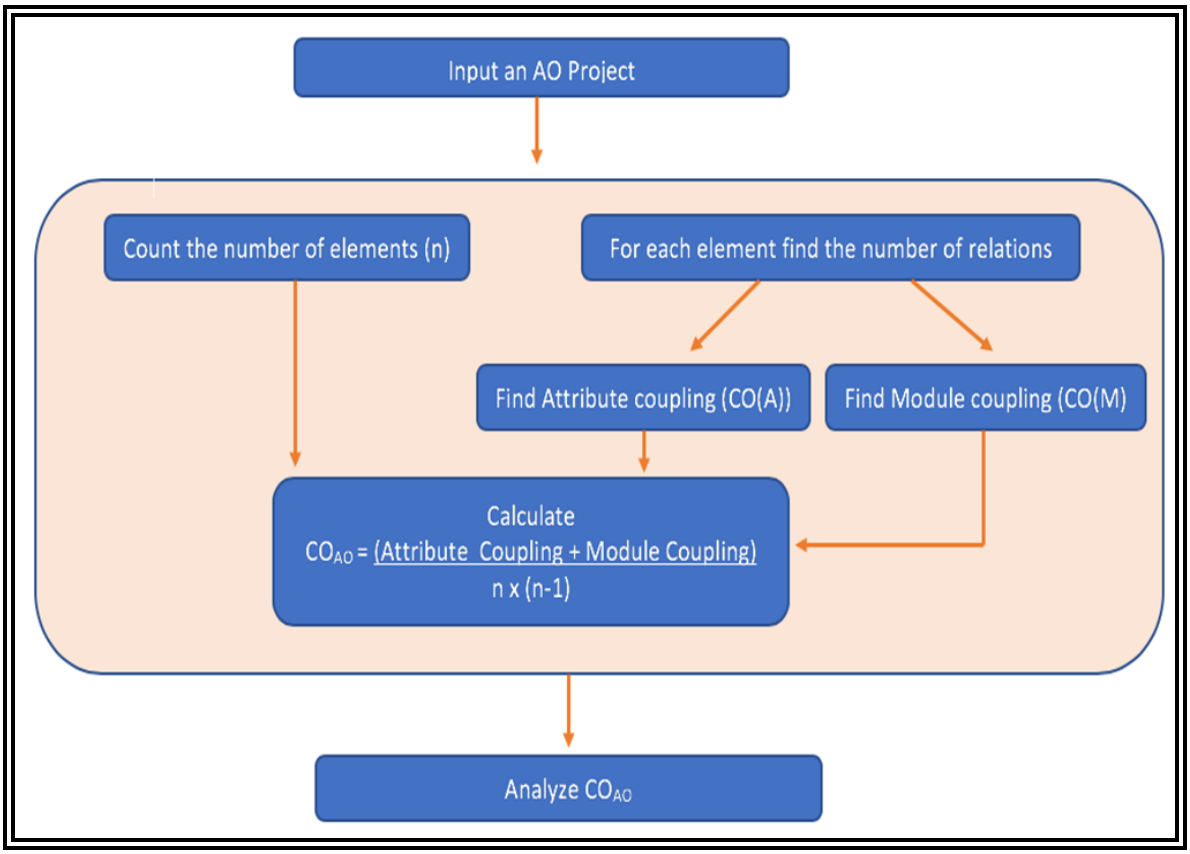


Figure 6.2: Methodology to measure and analyze the proposed aspect-oriented system coupling metric CO_{AO}

Module Coupling CO(M) is composed of elements that have been declared in the modules, which may possibly be triggered by a given element and then counting them. This includes the aspects that enclose advices that may be initiated along with the classes and interfaces that encompass methods that may be called. Mathematically, it may be stated, as shown in Equation 6.5.

For elements i and j in a set of elements E

there exists a module M(j), such that module M(i) →M(j) *(6.5)*

The value of CO_{AO} will range from zero to one, as described in Table 6.2. CO_{AO}= 1 indicates strong coupling, whereas CO_{AO}=0 indicates loose coupling.

Table 6.2: Qualitative categorization of Aspect-oriented Coupling

CO_{AO} Range	Coupling Category
0	No Coupling
$0 < CO_{AO} \leq 0.3$	Loosely Coupled
$0.3 < CO_{AO} \leq 0.7$	Average Coupled
$0.7 < CO_{AO} < 1$	Tightly Coupled
1	Highly Coupled

The proposed Aspect-oriented system coupling metric, CO_{AO}, is analyzed by considering the three cases that are presented below:

- ***Best case***

When all elements in an Aspect-oriented system are independent or when the system is empty, that is when there are no entities; then, there will not be any relation, and hence, $CO_{AO}=0$. That means, no coupling at all.

- ***Average case***

When approximately half of the elements in an Aspect-oriented System are related to one another, then the CO_{AO} will range from .3 to .7, indicating that 30 to 70 percent of the elements are connected via relationships to each other, and any change will cause a localized ripple effect.

- ***Worst case***

When all elements of the system are related to each other, then the number of relationships would be n (n-1); giving $CO_{AO} = 1$.

6.2.2.3 Illustration

For illustrating the proposed metric, an example aspect-oriented software with two packages, P1 and P2, as shown in Figure 6.3 is considered.

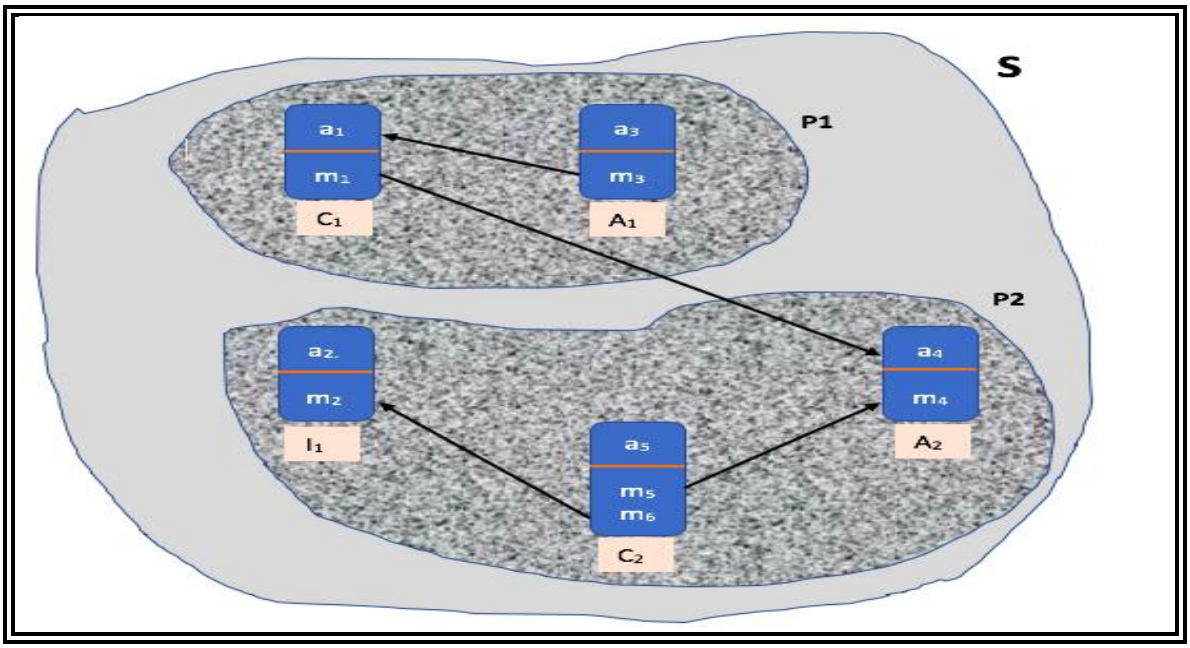


Figure 6.3: AOS Example

Using the example, as shown in Figure 6.3, the value of CO_{AO} is calculated by following the steps given for the evaluation of the Aspect-oriented system coupling metric, as illustrated in Figure 6.2. There is an AOS composed of a set of Elements in packages (P1 and P2) and a set of Relations. In Figure 6.3, there are five elements in total, including aspects (A1 and A2), classes (C1 and C2), and interfaces (I1). There are four relations, namely, r(A1,C1), r(C1,A2), r(C2,A2) and r(C2,I1).

Table 6.3: Identification of a Relation, as a contributor, to Attribute or Module Coupling

Relation	Type of Coupling
r(A1,C1)=1	Attribute Coupling
r(C1,A2)=1	Attribute Coupling
r(C2, A2)=1	Module Coupling
r(C2, I1)=1	Module Coupling

As per the identification and categorization in Table 6.3, we get

Attribute Coupling CO(A) =2 Module Coupling CO(M) =2 n=5

Using Equation 6.4, we get

$$CO_{AO} = \frac{CO(A)+CO(M)}{n*(n-1)} = \frac{2+2}{5*(5-1)} = \frac{4}{20} = 0.02 \quad (6.6)$$

Where CO_{AO}=0.02. Hence, the given Aspect-oriented System possesses loose coupling, as it falls in the range of $0 < CO_{AO} \leq 0.3$

Validation for the proposed AO System Coupling Metric is given in the next section.

6.2.3 AO Coupling Metric Validation

So as to ensure, that the proposed Aspect-oriented Coupling metric truly measures the Aspect-oriented software coupling characteristic, the theoretical soundness of the measure is checked based on the measures of the properties (internal characteristics) of the software. It is a prerequisite for metric acceptance and usage. There are various means for validating software engineering measurement [102]. The proposed Aspect-Oriented Coupling metric is theoretically validated using Property-Based software engineering measurements given by Briand et al. [101] for coupling, as presented in Table 6.4.

Table 6.4: List of Coupling Property Measures given by Briand

PROPERTY	COUPLING
Property 1	Non-negativity
Property 2	Null Value
Property 3	Monotonicity
Property 4	Merging of Modules
Property 5	Disjoint Module Additivity

6.2.3.1 Property 1- Nonnegativity

It states that the coupling of a system <E, R> is nonnegative.

The value of coupling of an Aspect-oriented System, as defined, will always be greater than or equal to zero. Hence, CO_{AO} satisfies Property 1.

6.2.3.2 Property 2- Null value

It states that the coupling of a system <E, R> is null if there are no relationships among entities of a system.

If a System S is empty, that is, if the Number of Elements E is zero ($E \in \Phi$) or Number of Relations R is zero ($R \in \Phi$), or both are null, the value of CO_{AO} will be zero. Hence, CO_{AO} satisfies Property 2.

6.2.3.3 Property 3- Monotonicity

It states that if a relation r(i , j) is added between element i and element j, then the value of coupling should not decrease.

The value of CO_{AO} will either be the same as the previous value or will increase with the addition of r(i , j). Hence, CO_{AO} satisfies Property 3.

6.2.3.4 Property 4- Merging of elements

It states that if two elements i and j are merged, then the value of coupling will decrease due to the inter-module relationship between the merged elements.

If $\exists$ r(i , j) $\in$ R, then the value of CO_{AO} will decrease because CO(A) or CO(M) or both will decrease. That is, upon merging of two elements, if there is any relation between the attributes or modules of the merging elements, then the value of CO(A) or CO(M) or both will decrease due to merging, thus, further reducing the value of CO_{AO}. Hence, CO_{AO} satisfies Property 4.

6.2.3.5 Property 5- Disjoint element additivity

It states that if two elements i and j are merged, then the value of coupling will remain unchanged due to the null relationship between the merging elements.

If r(i , j) ∉ R, then the value of CO_{AO} will remain unchanged as both CO(A) and CO(M) remain unaffected. That is, on merging two elements, if there is not any relationship between the attributes or modules of the merging elements, then the value of CO(A) or CO(M) will not be affected by merging and will remain the same, and the value of CO_{AO} will also remain unchanged. Hence, CO_{AO} satisfies Property 5.

As all Coupling properties 1 –5 hold by the CO_{AO} measure, it is a valid coupling measure for an Aspect-oriented System.

The detailed description of the proposed metric for the evaluation of supportability is given in the following section.

6.3 SUPPORTABILITY METRIC

Supportability metric measures the extent to which the software provides support and service for identifying and resolving issues while using the software or when it fails to work correctly. The metric for measuring supportability has been designed using the obtrusive data collection approach.

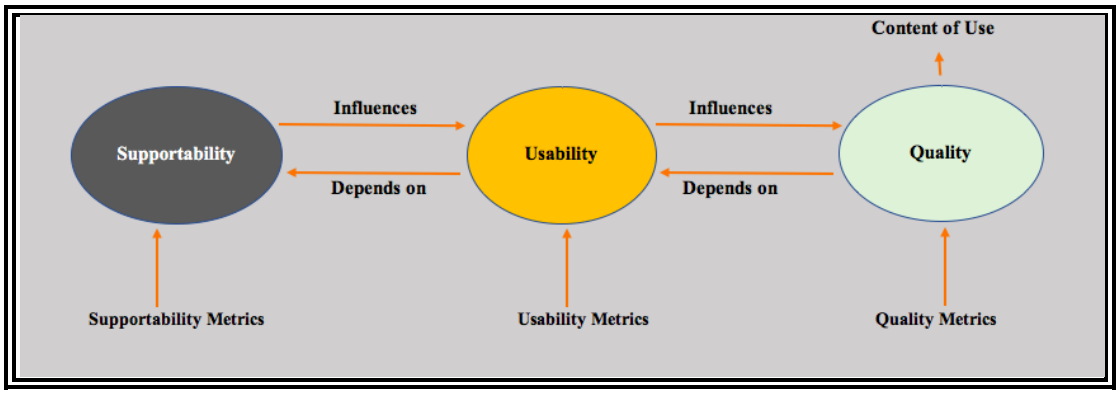


Figure 6.4: Relationship between Supportability and Quality

A sample questionnaire has been developed and proposed, which is to be filled by the end-user in an attempt to evaluate and assess the supportability provided by the software product. Also, it can be used to identify the weaknesses in the software product regarding supportability and identify the working areas which need improvement so as to ascertain as well as enhance the usability of the software

[103][104][105]. Relationship between quality and supportability is depicted in Figure 6.4.

The questionnaire contained statements about the comprehensive documentation, online support, usage of troubleshooting tools; help menus and event logging or tracing code, as shown in the (Appendix C Table A3.1). The central purpose of this sample questionnaire is to assess the supportability of software products. Evaluation of this will help in identifying any shortcomings with the aim to enhance supportability [106]. A similar questionnaire may be developed for the specific software product and its requirements — following which the methodology for evaluating the Supportability metric may be applied.

End Users are asked to identify the extent up to which they agree or disagree with each of the statements (Appendix C Table A3.1) in the range of 1 to 5, where 1 represents pre-dominantly disagree, 3 represents the average, and 5 represents pre-dominantly agree. 'SupportValue' is determined as the sum of the SValue evaluated using Equation 6.7 on the basis of the response of the user to the questions asked in the proposed questionnaire.

$$SupportValue = \sum_{i=1}^{n} \text{SValue(i)} \text{ where } 1 \leq \text{n} \leq 15 \qquad (6.7)$$

Where 'n' represents the number of questions in the questionnaire and 'SValue(i)' is the value corresponding to the response entered by the user in Question 'i'.

Thereafter, Supportability Metric is computed as per Equation 6.8.

$$Supportability\ Metric = \frac{SupportValue}{\text{n}*5} \times 10 \qquad (6.8)$$

Based on the inferences evaluated by Zuse [53], Supportability metric evaluated using equations (6.7) and (6.8), can further be analyzed using interval scale (in accordance to the standard statistics rules) as highlighted in Table 6.5.

Table 6.5: Supportability Reference Table

1-4	5-7	8-10
Needs Improvement	Satisfactory	Highly Supportable

Supportability is considered to be unsatisfactory if its value<=4 as that implies that most of the responses given by the users lie in the range of disagree. Supportability is considered to be satisfactory if its value lies in the range of 5 to 7. However, if the Supportability value is more than or equal to 8, then it is considered a highly supportable software.

6.3.1 Case Study

In an attempt to evaluate the proposed supportability metric, two industrial projects of academic level are selected that are interactive based application software. Project 1 is a text editor made under training program at CMC and Project 2 is quiz competition application software made under the training program at IQQUEST. Further details are available with the author on request. A set of evaluators were given the projects for usage and were asked to fill the supportability questionnaire as given in (Appendix C Table A3.1).

Table 6.6: Project 1 Supportability metric

	E1	E2	E3	E4	E5
Support Value	**34**	**29**	**21**	**30**	**27**
Supportability	**4.5**	**3.9**	**2.8**	**4.0**	**3.6**

Project 1 was assigned to five evaluators (Appendix C Table A3.2). Calculated values of the SupportValue and Supportability metric are given in Table 6.6.

Table 6.7: Project 2 Supportability metric

	E1	E2	E3	E4	E5
Support Value	**50**	**47**	**45**	**44**	**47**
Supportability	**6.7**	**6.3**	**6.0**	**5.9**	**6.3**

Project 2 was also assigned to five evaluators (Appendix C Table A3.3). SupportValue and Supportability metric values are shown in Table 6.7. Comparison of the supportability metric for the two projects is presented in Figure 6.5.

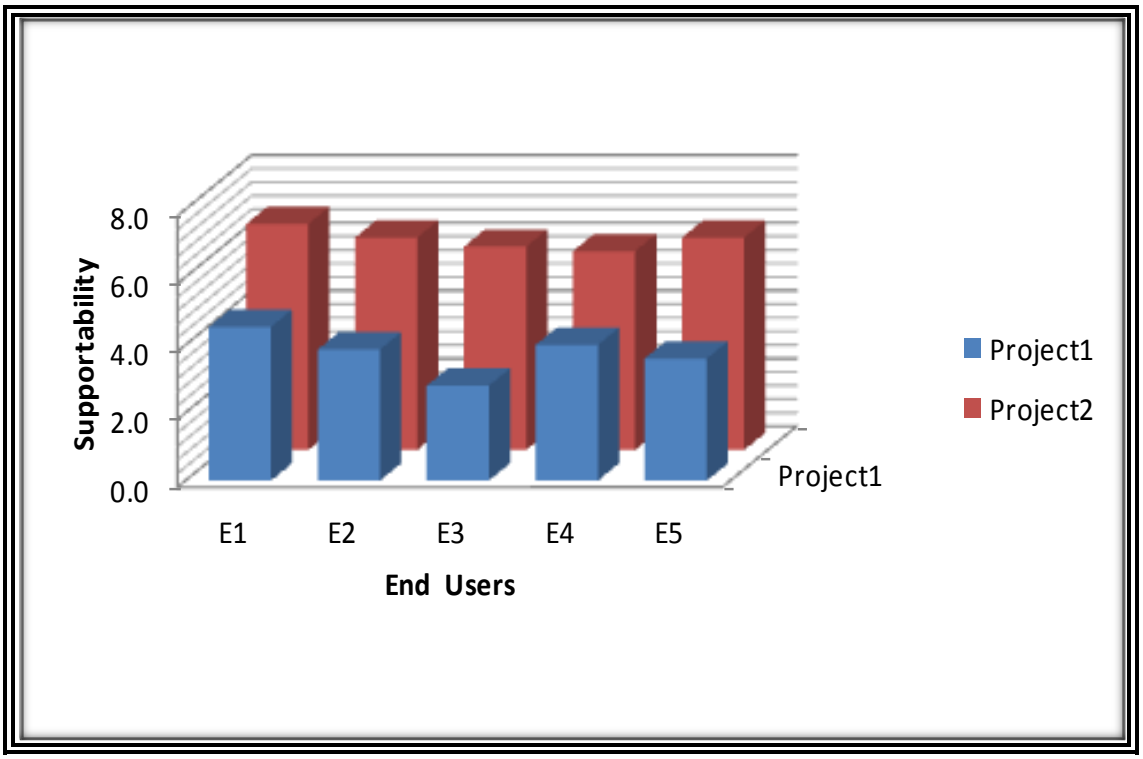


Figure 6.5: Supportability for Project1 and Project2

Average Supportability for Project1 and Project2 is found to be 3.8 and 6.2. Figure 6.5 evidently highlights that Project 2 is more Supportable than Project1 hence better adaptable and acceptable by the end-users than Project1.

The next section describes the proposed extensibility metric in detail.

6.4 EXTENSIBILITY METRIC

Taking into account that modern-day software applications involve management and processing of massive amount of data, originating from a multiplicity of sources such as the Internet of Things. Such data may be dynamic, homogenous, or heterogeneous. The users, according to their own specific needs, through programming code, process such data. The coding tool should be such that it is extensible and appropriate to manage such data. The aspect-oriented approach of programming can be a superior option to handle such data for extracting useful and timely information. The aspect-oriented approach can contribute positively to extending the software design and code dynamically. However, there is a need for a formal framework for evaluating the extensibility of the software.

To design and propose a framework for assessing extensibility, the required maintainability model, which was proposed and validated in previous Chapter 4, has been used. Overview of the proposed Extensibility framework is shown in Figure 6.6.

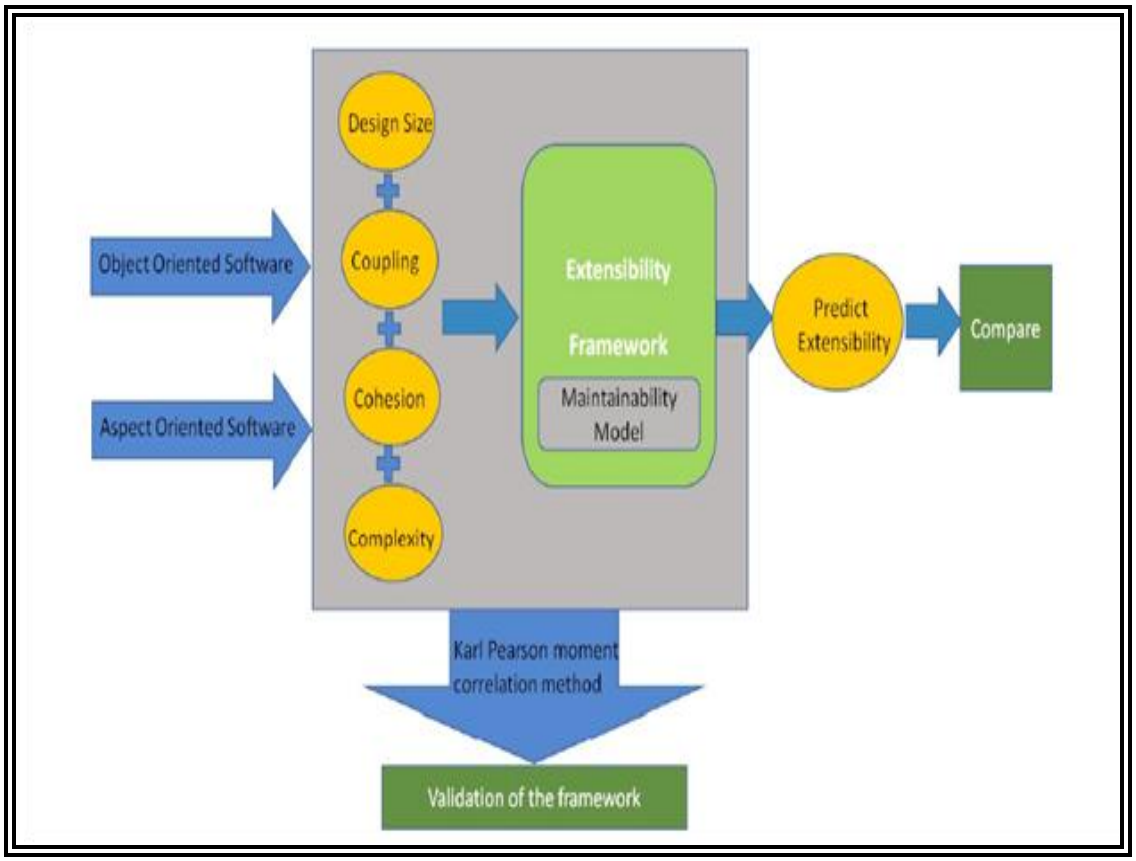


Figure 6.6: Overview of Extensibility Framework

6.4.1 Internal Factors and Metrics for Extensibility

Extensibility is the capability of software to append functionality with no diverse effect on the system. Functionality changes may occur due to altering or enhancing requirement specifications. Extensibility may be considered a specific type of reuse of an element. It is a standardized measure of the ability to broaden a software design to incorporate the implementation while considering future growth. The extensible design avoids software development issues such as low cohesion and high coupling. Figure 6.7 depicts the relationship between extensibility quality characteristics with internal factors and metrics.

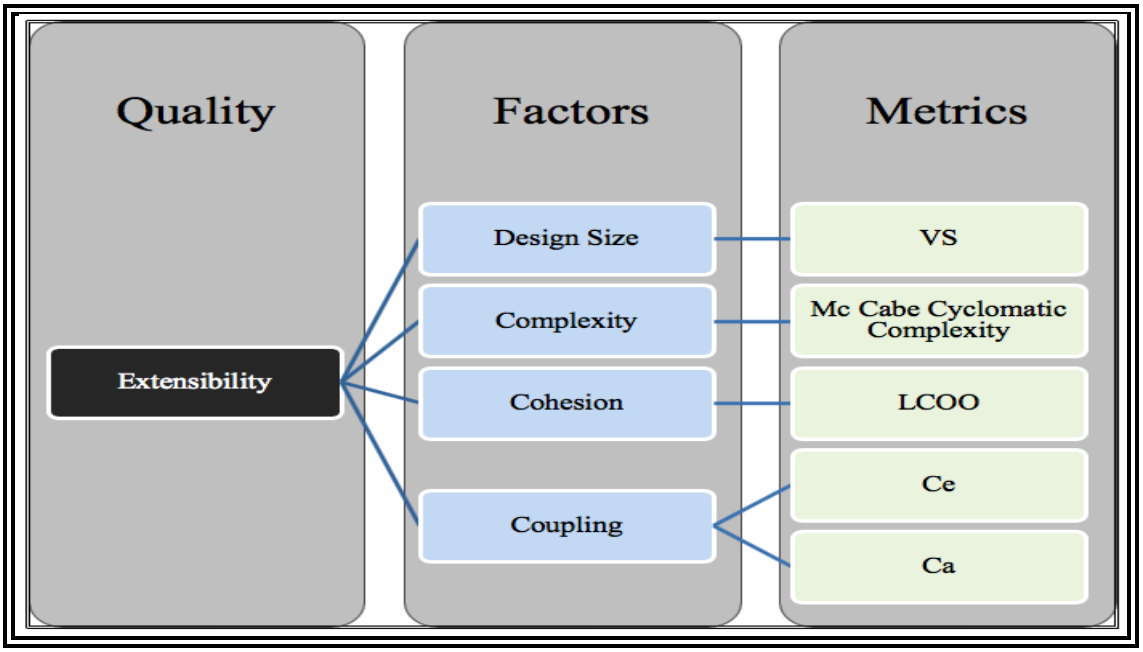


Figure 6.7: Relationship of extensibility quality characteristics with internal factors and metrics

- **Design size**

Software design considers modularity as one of the fundamental principles in software engineering. It works on the principle of dividing a complex system into simpler pieces called modules. The division is based on the separation of concerns. Modularization reduces the complexity of the software and improves maintainability, extensibility, and productivity. It is computed by counting the number of classes, interfaces, and aspects in the AO software.

- **Complexity**

Code complexity is a measurement relative to coding errors. To obtain, high-quality software, with minimal testing and maintenance cost, code complexity must be checked regularly. It is related to the number of decision points present in the code. This identification assists in locating the hidden knots of logic in the code. Furthermore, the identified highly complex code sections might be bifurcated into smaller, manageable, and logical sub-sections. It is computed by using the Mc Cabe Cyclomatic complexity metric.

- **Coupling**

Coupling is the measure of the strength of the interconnection between modules and assesses the type and number of interconnections among modules. Although coupling cannot be zeroed, it can be brought to controllable levels with the use of modularity and appropriate encapsulation. It is computed by assessing afferent and efferent coupling in the AO software.

- **Cohesion**

Cohesion is a positive aspect of the module. It is the association among the different components of a module and assesses why the components are grouped together in a module. It is computed by determining the lack of cohesion of methods metric.

Table 6.8: Metrics for Design Characteristics

Design Characteristic	Design Metric
Design Size(DS)	Number of classes, interface, and aspects
Complexity(CO)	McCabe cyclomatic complexity metric
Cohesion(Coh)	Lack of cohesion of methods
Coupling(Cou)	Total (efferent and afferent) coupling

The metrics used to collect information regarding the various design characteristics related to the extensibility of AOS are given in Table 6.8. Using these metrics, an assessment of extensibility is made for a set of AOS software.

6.4.2 Proposed Extensibility Metric

In an attempt to measure the qualities of aspect-oriented software precisely, the related attributes for each quality need to be selected [107]. In this work, the focus is on a specific type of reuse of a component called extensibility, i.e., the extension of software without accessing existing code to edit or copy it [107]. The proposed metric is formulated based on the weighting method to measure extensibility.

Extensibility is a systemic measure of the ability to extend a software design principle [107], which is derived by using the following condition shown in Equation 6.9.

$$Extensibility = 0.25 * DS + 0.25 * CO + 0.50 * CC \quad (6.9)$$

Where,

DS refers to the design size,

CO refers to the complexity, and

CC refers to the cohesion/coupling.

6.4.3 Case Study

The proposed framework for extensibility is tested for a set of aspect-based software. Set of software; built-in AspectJ; are selected, and extensibility is computed using the attributes identified in the previous section. Summary of the metrics collected for the list of AspectJ projects is given in Table 6.9.

Table 6.9: List of AspectJ Projects

AspectJ Project	Number of packages	Lines of code
Spacewar AspectJ	2	1415
Bean AspectJ	1	123
Introduction AspectJ	1	71
Observer AspectJ	1	128
Telecom AspectJ	1	181
TJP AspectJ	1	49
Tracing AspectJ	1	84
AJHotDraw	24	21564

Table 6.10 gives the value of extensibility for all AspectJ software listed in Table 6.9 calculated by applying the metrics defined in Table 6.8 and Equation 6.9. [As given in Appendix B Figure A2.3, Figure A2.4 and Appendix D Figure A4.1 – Figure A4.7]

Table 6.10: Extensibility of AspectJ Projects

	Design Size	Complexity	Cohesion	Coupling	Extensibility
Spacewar AspectJ	29	1.934	0.704	2	**7.91**
Bean AspectJ	3	1.182	0.25	1	**1.17**
Introduction AspectJ	4	1.167	0.63	1	**1.61**
Observer AspectJ	8	1	0	1	**2.25**
Telecom AspectJ	13	1.194	0.286	1	**3.69**
TJP AspectJ	2	1	0	1	**0.75**
Tracing AspectJ	6	1	0.062	1	**1.78**
AJHotDraw	381	1.592	.737	10.333	**95.684**

In order to make a comparable analysis, first seven projects are used to ascertain the relationship between the four selected attributes (design size, complexity, cohesion, coupling) and extensibility as shown in Figures 6.7, 6.8, 6.9, and 6.10, respectively.

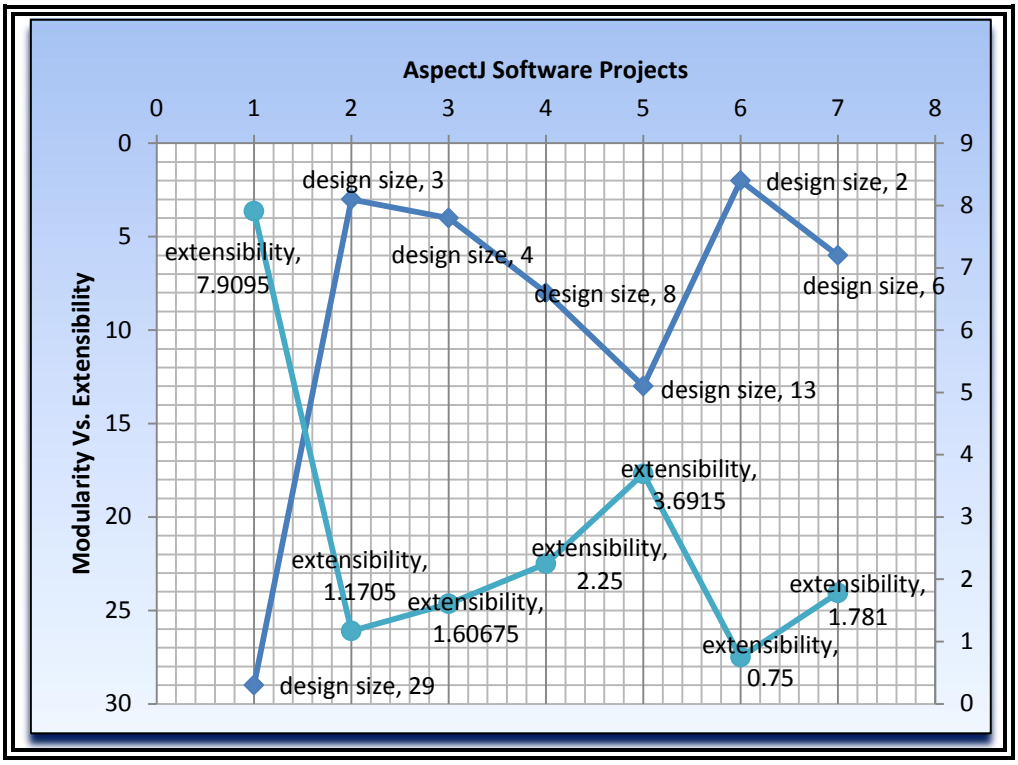


Figure 6.8: Relation between Design Size and Extensibility

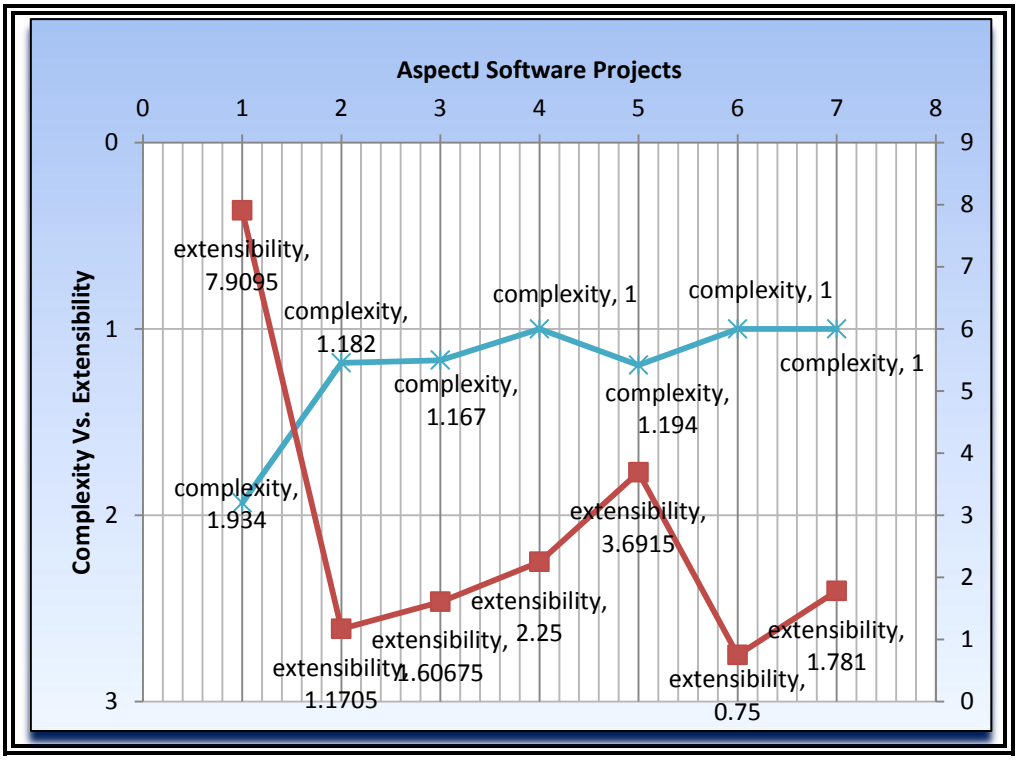


Figure 6.9: Relation between Complexity and Extensibility

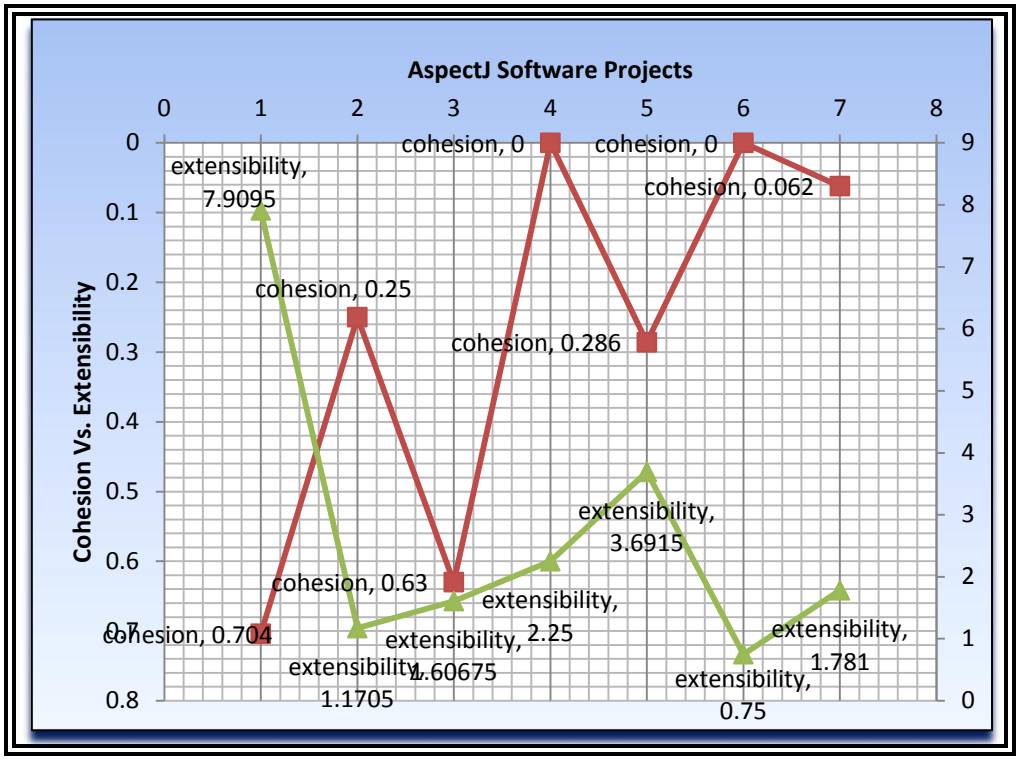


Figure 6.10: Relation between Cohesion and Extensibility

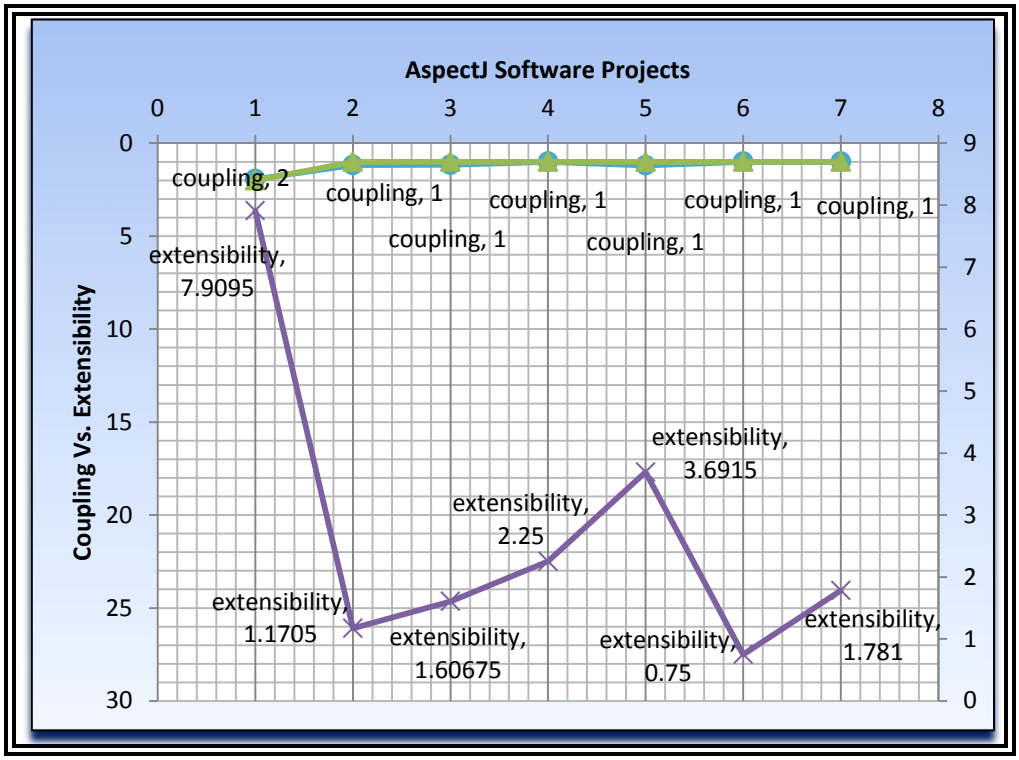


Figure 6.11: Relation between Coupling and Extensibility

Figure 6.8 to 6.11 show the relationship between the attributes and extensibility. As depicted in Figure 6.8, even a small change in design size is reflected strongly in extensibility. Extensibility varies with even a small change in design size; which gives the feeling of strong interrelation between design size and extensibility. Extensibility increases with an increase in design size and extensibility decreases with a decrease in design size. Variation in complexity also affects the extensibility measure, as reflected in Figure 6.9. The results confirm the strong correlation between the complications of the software with its extensibility. Figure 6.10 depicts the effect of cohesion between the software systems on its extensibility. Increase or decrease in cohesion is reflected direct proportional in extensibility. Extensibility variance with the change in coupling is depicted in Figure 6.11. To further add confidence in the given extensibility metric, a correlation analysis is done in the following subsection.

6.4.4 Extensibility Metric Validation

To ensure that the proposed extensibility metric measures the extensibility characteristic of the aspect-oriented software using the modularity, complexity, cohesion, and coupling metrics, a correlation between the sub-attributes and extensibility must be established. The Karl Pearson Product Moment correlation technique is used to find the correlation value. The computed values are shown in Table 6.11.

Table 6.11: Correlation values for DS, CO, CC and Extensibility

Attributes	Design Size	Complexity	Cohesion	Coupling
Extensibility	0.99	0.90	0.68	0.81

The findings in Table 6.11 reveal a strong positive relationship between the selected attributes and extensibility. Thus, it can be concluded that the level of sub-attributes can contribute effectively to determine the level of extensibility, with design size being the top contributor, followed by complexity, coupling, and cohesion.

6.4.5 Extensibility Framework Comparison for OO and AO Software

In order to ensure the novelty of the extensibility framework, the extensibility for the existing object-oriented software development approach is compared with the latest aspect-oriented software development approach.

The Spacewar Project is selected to compare extensibility for object-oriented and aspect-oriented software. For object-oriented, the Spacewar code is built in Java, and for aspect-oriented, the Spacewar code is built in AspectJ. The lines of code in Java and AspectJ are 1991 and 1415, respectively. The Spacewar Java code is composed of 22 classes and interfaces, while the Spacewar AspectJ code is composed of 29 classes, interfaces, and aspects.

The comparison graph in Figure 6.12 illustrates the extensibility measure of the Spacewar software implemented in Java and AspectJ.

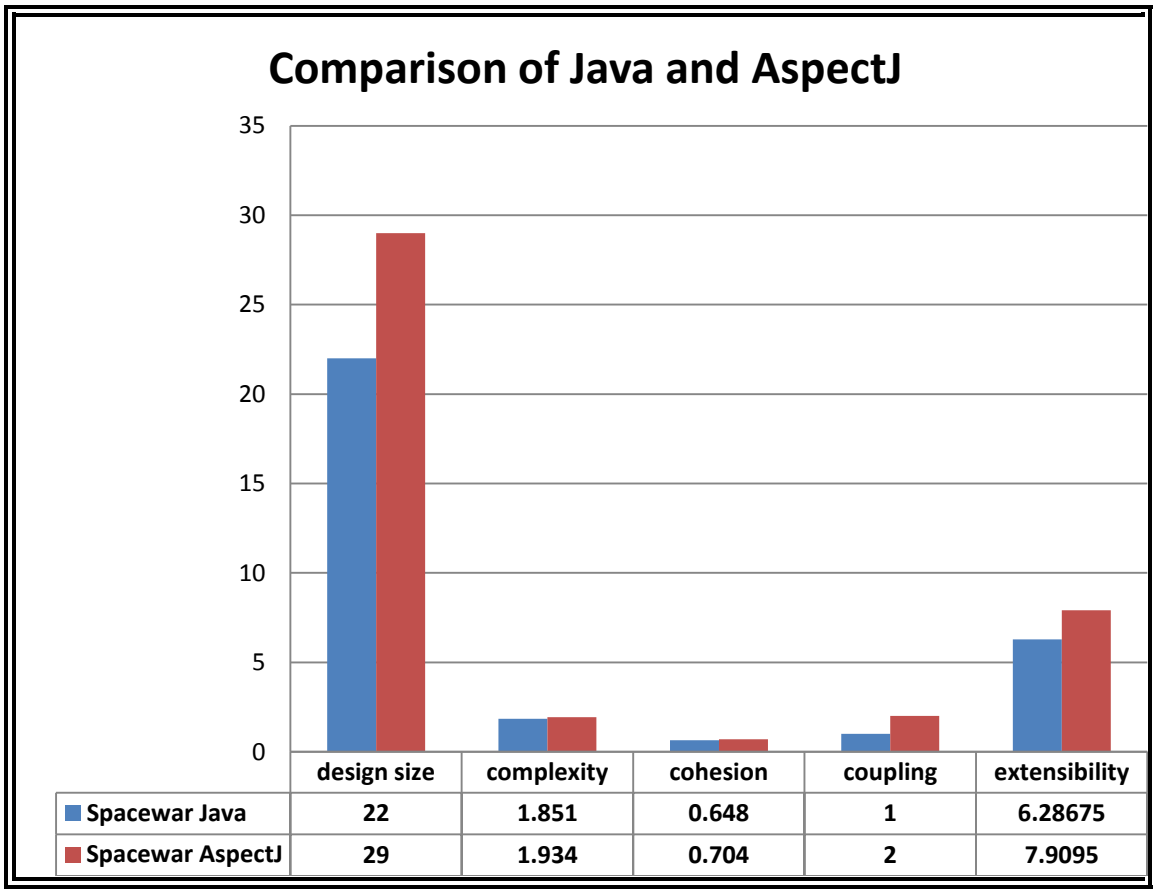


Figure 6.12: Extensibility analysis

The results show that the extensibility of the Spacewar software build in Java is calculated as 6.3 (approx.), while that of the software build in AspectJ is 7.9 (approx.). Therefore, it is easier to extend the functionality of the same project built in AspectJ than in Java.

The next chapter concludes the outcome of the work proposed in this thesis. The future research directions are also enumerated in this regard.

CHAPTER VII

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

With time and evolution, quality has become a mandatory attribute of any category of software. On a daily basis, a significant number of software products are developed and released by the IT industry. Software quality models are used as a vital tool for quality assessment and assurance. Many software quality standards and measure have been used by the software industry personnel so as to ensure and improve the quality of the software. But still there exist many untouched areas of prime interest that can significantly improve the quality of the software, its measurement, and analysis. Towards this goal, an extensive survey on Quality aspects has been done keeping the focus on three main software programming paradigms, namely; module oriented programming, object-oriented programming, and aspect-oriented programming. Survey details identified various quality models and software metrics for measuring the quality of the software. Critical analysis of all the reviewed aspects of quality-related to programming methodology being used is done, and significant challenges towards measuring, analyzing software quality were identified that further become the basis for objectives of work carried out in this dissertation.

The foremost objective of the work is to build a software quality model that can incorporate modern programming features and attribute to improve overall software quality assessment. To achieve this objective, a valid software quality model has been developed on the lines of ISO 25010. The contribution made by the present work are listed below:

- **An Improved Model To Estimate Quality Of The Software Product**

The existing software quality models have been critically studied, and a relative comparison among them is made. After investigating them in detail and examining their limitations, a modified software quality model based on the guidelines of latest

ISO/IEC 25010, is being proposed, with a view of taking into consideration the present-day complexities and requirements of software products.

- **Quantitative Evaluation of Proposed Maintainability Model using AHP Method**

This work aims to validate and evaluate the proposed Aspect-oriented software maintainability Quality model as a single unit using the analytic hierarchy process (AHP). This model has considered six attributes, namely extensibility; reusability; modifiability; analysability; testability and modularity concerning maintainability characteristic of quality. To conduct the AHP technique, the surveys on participants from the IT industry have been carried out, and the value of pairwise relative weights for the characteristics is taken. The mean of the collected samples has been considered as pairwise relative weights. The case study validates the suitability and the usefulness of the proposed model. The final computed Eigenvector gives the relative ranking of quality attributes in relation to maintainability in the order of extensibility; reusability; modifiability; analysability; testability and modularity.

- **Analysis of Reliability Model with the Application of MCDM**

To analyze and evaluate the reliability characteristic of the software product or project, initially, various predefined models for the software quality concerning reliability attribute in them as particular are reviewed. Through analysis, it was diagnosed that although all the attributes relative to reliability are included in the ISO 25010 model other than scalability. Hence, a new model for software reliability with the availability, maturity, fault tolerance, recoverability along with scalability is proposed. With the objective to confirm the consistency of the proposed model, the survey among the software industry people is conducted. Participants from various reputed software industry participated in the survey. AHP method is applied for ensuring the consistency of the proposed reliability model. Evaluated results authenticate that the chosen sub characteristics for the software reliability are consistent. The relative ranking of the sub characteristics is scalability, maturity, fault tolerance, recoverability, and then availability.

- **Performance Efficiency Assessment for Software Systems**

To assess the performance efficiency characteristic of the software systems, firstly the criterion/ factors that could affect the performance efficiency are identified and structured into levels, and the performance efficiency model is proposed. AHP method for ensuring the consistency of the proposed performance efficiency model is applied. In order to assess and validate the proposed model, a survey is conducted in which participants from software industry background participated. After AHP evaluation, results demonstrate that the chosen quality sub characteristics are consistent and the relative ranking of the quality attributes for performance efficiency are in the order of time behavior; optimized code; resource utilization and then capacity.

- **Incorporating Supportability in Software Usability and its Assessment**

As software product supportability is an essential feature for improving quality; hence in this work suggests Supportability characteristic needs to be included as an additional attribute to the Usability quality feature. Calculated results, after the application AHP technique, proved the new proposed usability model to be consistent. The relative ranking of the factors is evaluated in the order of appropriateness recognizability, learnability, supportability, operability, user error protection, user interface aesthetics, and accessibility. Also, a metric for measuring supportability has been designed using the obtrusive data collection approach.

- **Critical Assessment of Aspect Orientation Metrics and Quality**

In order to investigate, which software metrics are helpful for assessing the quality of aspect-oriented software, a systematic investigation is conducted and hence analyzed the relationship of the AOP metrics with quality. Nearly sixty-five AOP metrics based on aspects, joinpoints, pointcuts, introductions, etc. for aspect-oriented programming approach have been analyzed along with their connectivity with the overall software quality. Also, during the critical examination of the metrics, various metrics are identified to affect precisely complexity, extensibility, reusability, encapsulation, and understandability of the Aspect-oriented Software.

- **Investigation of Reusability and Complexity of AOP systems**

The Quality of Aspect-Oriented Software system is expected to improve the quality of the developed software products by separating the non-functional concern from the core concern. To analyze the impact, the OO metrics for the Spacewar software made in Aspect-oriented language AspectJ and Object-oriented language Java, both, are collected and compared. From the 23 metrics collected, the results obtained for AspectJ software are significantly better than its Java counterpart in terms of modularity, stability, maintainability, and extensibility, however at the cost of complexity. This work uses a set of metrics defined initially for Object-oriented systems. The metrics may be biased for OOP as they were created in context for OOP before the advent of AOP. This comparison will work as a stepping stone for assessing the quality of the software in terms of reusability, maintainability, and complexity.

- **Aspect-oriented system coupling metric and its validation**

Coupling Metric for the complete aspect-oriented system is proposed in this work. For this, first, a literature review is performed to review the current status of the metrics in aspect-oriented programming. Particular emphasis is given on assessing the coupling in aspect-oriented software systems. It was identified that although metrics exist for evaluating the coupling of aspect-oriented systems, they are at the basic levels of fields, methods, classes, or are aspects as standalone entities. Only a few metrics exist for the measurement of software quality attributes at a higher level of abstraction in AO systems. Hence, there is a need for one metric for complete Aspect-oriented Software System Coupling. To accomplish this, formal definitions of the terminology used concerning aspect-oriented programming are proposed. On the basis of the clearly defined definitions, a formal mathematical metric is proposed for measuring the coupling of an aspect-oriented system, namely, the Aspect-Oriented System Coupling Metric CO_{AO}.

Further, an illustration of an example aspect-oriented system is made to demonstrate the calculation of the proposed metric. To ensure the accuracy and enhance the confidence in this metric, the proposed metric is validated against the five property measures of coupling for software engineering that was projected by Briand [101]. As

all Coupling Properties 1 –5 hold by CO_{AO} measure, hence, the proposed aspect-oriented coupling metric CO_{AO} is acceptable as a valid coupling measure for an Aspect-oriented System.

- **A Framework For Evaluating Extensibility In An Aspect-Oriented Software System And Its Validation**

The framework for evaluating the extensibility of aspect-oriented systems is presented in this work. To accomplish this, a maintainability model for aspect-oriented software systems that are proposed earlier has been used, and a novel framework for evaluating the extensibility characteristic is formulated.

Further, testing of the proposed extensibility metric has been performed by demonstrating the calculation of the proposed metric using a set of software projects developed in AspectJ. Also, Karl Pearson Product Moment Correlation method is used for validating the proposed extensibility metric. The results show that the metric for measuring extensibility is appropriate.

Finally, a comparison has been performed for the software project built in OOP and AOP in relation to extensibility. The analysis done indicates that the software project built using AOP approach is more extensible than the one built using OO approach.

7.2 FUTURE SCOPE

In this work, various issues related to aspect-oriented quality model and measurement have been addressed. But there is still a scope of improvement in a few areas that are worth exploring for providing useful, specific, and timely information in the form of metrics and measurements to software products. The list of some of these issues ranging from existing software quality model, improvement to dealing with associated metrics that are largely ignored by the current quality models is given below:

- The software metrics identified to be strongly affecting the trustworthiness of the software can be used to measure the trustworthiness of the various software products.

- The proposed maintainability, reliability, performance efficiency, and usability model can be applied and cross-validated for other than Aspect-oriented approaches and can be used for the relative comparison of the attributes.

- Further, the proposed model may be used to diagnose the specific software weakness in terms of usage easiness and the acceptance of the customer.

- In the future, other external attributes such as extensibility, performance efficiency, maintainability, and testability can be assessed and analyzed using the proposed aspect-oriented coupling metric for Aspect-oriented systems.

- Autonomic features of a software product may also be incorporated.

- Henceforward, the proposed quality framework can be used to analyze total software quality for aspect-oriented systems. The application of agile modeling approach can also be investigated.

REFERENCES

[1] IEEE Standard Glossary of Software Engineering Terminology, *In: IEEE Std 610.12-1990*, 1990, pp.1-84. DOI: 10.1109/IEEESTD.1990.101064.

[2] Roger S. Pressman, Software Engineering: A Practitioner's Approach (5th edition), New York: McGraw-Hill Higher Education, ISBN: 0072496681, 2001.

[3] A. Kaur, P.S. Grover, A. Dixit, "Performance Efficiency Assessment for Software Systems", *In Software Engineering, Advances in Intelligent Systems and Computing*, vol.: 731, Singapore: Springer, pp. 83-92, 2019.

[4] Ian Sommerville; Software Engineering; 9th edition, Addison-Wesley, USA, ISBN-13: 978013703515-1, 2010.

[5] G. Kiczales et al., "Aspect-Oriented Programming", *In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97),* Springer, pp. 220-242, 1997.

[6] G. Kiczales et al., "An Overview of AspectJ", *In: Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01),* Springer, pp. 327-353, 2001.

[7] A. Kaur, P. S. Grover and A. Dixit, "Analysis of Quality Attribute and Metrics of various software development methodologies", *In: Proceedings of International Conference on Advancements in Computer Applications and Software Engineering*, pp. 05 - 10, 2012.

[8] A. Kaur, P. S. Grover and A. Dixit, "An Improved model to estimate Quality of the Software Product", *YMCAUST International Journal of Research*, vol.: 1, issue: 2, pp. 01 - 06, 2013.

[9] M. S. Ali et al., "A systematic review of comparative evidence of aspect-oriented programming", *Information and Software Technology*, vol.: 52, issue: 9, pp. 871-887, 2010.

[10] K. Sirbi and P. J. Kulkarni, "On Using Metrics in Evaluation of Aspect Oriented Programming Maintainability -A Research Article", *International*

Journal Empirical Software Engineering, 2013

[11] A. Kumar, R. Kumar and P.S Grover, "An evaluation of maintainability of aspect oriented systems: A practical approach", *International Journal of Computer Science and Security*, vol.:1, issue: 2, pp. 1-9, 2007.

[12] T. Li et al. "A storage solution for massive IoT data based on NoSQL" *In IEEE Proceedings of International Conference of Green Computing and Communications (GreenCom)*, pp. 50-57, 2012.

[13] T. Spieldenner et al., "FiVES: An Aspect-Oriented Virtual Environment Server", *In IEEE Proceedings of International Conference on Cyberworlds (CW)*, pp. 103-110, 2017.

[14] S. Becker et al., "Trustworthy software systems: a discussion of basic concepts and terminology", *ACM SIGSOFT Software Engineering Notes*, vol.:31, issue: 6, pp. 1-18, 2006.

[15] M. Pinto, J. M. Horcas, "How to develop secure applications with Aspect Oriented Programming", *In IEEE Proceedings of International Conference on Risks and Security of Internet and Systems (CRiSIS)*, pp. 1-3, 2013.

[16] V. O. Safonov, "Using aspect-oriented programming for trustworthy software development", Wiley-Interscience Publication, vol.: 5, 2008.

[17] A. Kaur, P. S. Grover and A. Dixit, "Critical Assessment of Aspect Orientation Metrics and Quality", *Recent Trends in Programming Languages (RPTL)*, vol.: 5, issue: 2, pp. 15-23, ISSN- 2455-1821, 2018.

[18] A. Kaur, P. S. Grover and A. Dixit, "Investigation of Reusability & Complexity of AOP systems", *International Journal of Innovations & Advancement in Computer Science (IJIACS)*, vol.: 7, issue: 3, pp.1-5, ISSN-2347-8616, 2018.

[19] T.L. Saaty, Analytic Hierarchy Process, Encyclopaedia of Biostatistics, vol.:1, 2005.

[20] A. Kaur, P. S. Grover and A. Dixit, "Incorporating Supportability in Software Usability and its Assessment", Communicated in *Indian Journal*

of Pure and Applied Mathematics, ISSN: 0019-5588, April 2019.

[21] A. Kaur, P. S. Grover and A. Dixit, "A framework for evaluating extensibility in an Aspect-oriented software system and its validation", *Recent Patents on Engineering*, vol.: 13, ISSN: 2212-4047, 2019. DOI: 10.2174/1872212113666190625115111

[22] A. Kaur, P. S. Grover and A. Dixit, "Aspect-oriented system coupling metric and its validation", *Recent Patents on Computer Science*, vol.: 12, ISSN: 1874-4796, 2019. DOI: 10.2174/2213275912666190410143540

[23] J. A. McCall, P. K. Richards and G. F. Walters, "Factors in Software Quality", Volumes I, II, and III. US Rome Air Development Center Reports, US Department of Commerce, USA, 1977.

[24] Marc-Alexis Côté, Witold Suryn and Elli Georgiadou, "Software Quality Model Requirements for Software Quality Engineering", *In Proceedings of 14th International Software Quality Management & INSPIRE Conference (SQM),* Southampton Hampshire, UK, 2006.

[25] B. W. Boehm, J. R. Brown, M. Lipow, "Quantitative evaluation of software quality", *In IEEE Proceedings of the 2nd International Conference on Software Engineering,* Los Alamitos (CA), USA, pp.592-605, 1976.

[26] B. W. Boehm et al., "Characteristics of Software Quality", North-Holland Publishing, 2nd edition, Amsterdam, The Netherlands, 1978.

[27] ISO 2001; ISO/IEC 9126-1; Software Engineering –product quality- Part 1, Quality model, International Organization for Standardization, Geneva; Switzerland, 2001

[28] R. E. Al-Qutaish; "An Investigation of the Weaknesses of the ISO 9126 International Standard"; *In Proceedings of Second International Conference on Computer and Electrical Engineering*, Dubai, vol.: 1, pp. 275-279, 2009. DOI:10.1109/ICCEE.2009.83

[29] C. Wohlin, M. Ahlgren, "Soft factors and their impact on time to market", *Software Quality Journal*, vol.: 4, issue: 3, pp. 189-205, 1995.

[30] ISO/IEC 2011; ISO/IEC 25010:2011, Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models.

[31] S. Commander, "The software industry in emerging markets", Edward Elgar Publishing, eISBN: 9781781958513, 2005.

[32] N. Fenton, J. Bieman, "Software metrics: a rigorous and practical approach", Third Edition, CRC press, USA, 2014.

[33] B. Mehndiratta, P. S. Grover, "Software metrics—an experimental analysis", *In Newsletter ACM SIGPLAN Notices*, vol.: 25, issue: 2, pp. 35 - 41, 1990.
DOI=10.1145/96429.96435

[34] A. Shaik et al., "Metrics for Object Oriented Design Software Systems: A Survey", *Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS)*", vol.: 1, issue: 2, pp. 190-198, 2010.

[35] S.R. Chidamber, C.F. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol.: 20, issue: 6, pp. 476-493, 1994.

[36] M. Lorenz and J. Kidd, "Object-Oriented Software Metrics", Prentice-Hall, USA, 1994.

[37] J.Zhao and B. Xu, "Measuring aspect cohesion", *In Proceedings of International Conference on Fundamental Approaches to Software Engineering (FASE)*, Springer, Berlin, Heidelberg, pp. 54-68, 2004.

[38] J.Zhao, "Measuring coupling in aspect-oriented systems", *In Proceedings of the 10th International Software Metrics Symposium (METRICS'04)*, pp. 1-9, 2004.

[39] M. Ceccato and P. Tonella; "Measuring the effects of software Aspectization", *In Proceedings of the 1st workshop in Aspect Reverse Engineering*, vol.: 12, 2004.

[40] M. Bartsch and R. Harrison, "An exploratory study of the effect of Aspect Oriented Programming on maintainability", *Software Quality Journal*, vol.: 16, pp. 23-44, 2008.

[41] L. Herrejon, E. Roberto and S. Apel: "Measuring and Characterising cross cutting in Aspect base programs; basic metrics and case studies", *In Proceedings of International Conference on Fundamental Approaches to Software Engineering (FASE)*, Springer, Berlin, Heidelberg, pp. 423-437, 2007.

[42] R. Burrows, A. Garcia, F. Taïani, "Coupling metrics for Aspect Oriented Programming: A systematic review of maintainability studies", *In Proceedings of Evaluation of Novel Approaches to Software Engineering (ENASE)*, Springer, Berlin, Heidelberg, pp. 277-290, 2009.

[43] C. Sant'Anna et al., "On the reuse and maintenance of aspect-oriented software: An assessment frame-work", *In Proceedings of XVII Brazilian Symposium on Software Engineering*, pp. 19-34, 2003.

[44] C. Sant'Anna et al., "On the Modularity Assessment of Software Architectures: Do my architectural concerns count?" *In proceedings of International Workshop on Aspects in Architecture Descriptions (AARCH. 07), AOSD'2007*, Vancouver, British Columbia, vol.: 7, 2007.

[45] A. Kumar; "Analysis and design of metrics for Aspect Oriented Systems", Doctoral dissertation, Thapar University, 2010

[46] P. S. Grover, R. Kumar and A. Kumar, "Measuring Changeability for Generic Aspect–Oriented Systems", *ACM SIGSOFT Software Engineering Notes*, vol.: 33, issue: 6, pp. 1-5, 2008.

[47] A. Kumar, R. K. Bhatia and P. S. Grover, "Generalized coupling measure for aspect-oriented systems" *ACM SIGSOFT Software Engineering Notes* , vol.: 34, issue: 3, pp. 1-6, 2009.

[48] A. Kumar, R. Kumar and P. S. Grover, "Unified Cohesion Measures for Aspect-Oriented Systems", *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 21, issue: 1, pp. 143-163, 2011

[49] R. Kumar, P. S. Grover and A. Kumar, "A Fuzzy Logic Approach to Measure Complexity of Generic Aspect-Oriented Systems", *Journal of Object Technology (JOT)*, vol.: 9, issue: 3, pp. 59-77, 2010.

[50] A. Kumar, P. S. Grover and R. Kumar, "A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO)", *ACM SIGSOFT Software Engineering Notes,* vol.: 34, issue: 5, pp. 1-9, 2009.

[51] M. Speicher, "What is usability? A characterization based on ISO 9241-11 and ISO/IEC 25010", arXiv preprint arXiv: 1502.06792, 2015.

[52] A. Seffah et al., "Usability measurement and metrics: A consolidated model", *Software Quality Journal*, vol.: 14, issue: 2, pp. 159-178, 2006. DOI=10.1007/s11219-006-7600-8

[53] H. Zuse, "Properties of software measures", *Software Quality Journal,* vol.: 1, issue: 4, pp. 225-260, 1992. DOI=10.1007/BF01885772

[54] J. Nielsen, "Usability engineering", *Elsevier,* 1994.

[55] H. J. Lee et al., "A User eXperience Evaluation Framework for Mobile Usability", *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)",* vol.: 27, issue: 2, pp. 235-79, 2017.

[56] D. Fontdevila, M. Genero and A. Oliveros, "Towards a usability model for software development process and practice", *In Proceedings of International Conference on Product-Focused Software Process Improvement*, Springer, Cham, pp. 137-145, 2017.

[57] A. H. M. Katy, "Measuring usability for application software using the quality in use integration measurement model", Doctoral dissertation, Universiti Tun Hussein Onn, Malaysia, 2016.

[58] R. B. Grady and D. Caswell, "Software Metrics: Establishing a Company-wide Program", Prentice Hall, USA, 1987

[59] S. K. Dubey, S. Ghosh and A. Rana, "Comparison of software quality Models: An Analytical Approach", *International Journal of Emerging Technology and Advanced Engineering (IJETAE),* vol.: 2, issue: 2, pp. 111-119, 2012.

[60] C. Ghezzi, M. Jazayeri, and D. Mandrioli, "Fundamental of software Engineering", Prentice Hall, NJ, USA, 2002.

[61] K. Khosravi, Y. G. Gueheneuc, “On Issues with Software Quality Models”, *In Proceedings of the 11th Working Conference on Reverse Engineering*, pp. 172-181, 2004.

[62] F. Khomh, “SQUAD: Software Quality Understanding through the Analysis of Design”, *In Proceedings of the 16th Working Conference on Reverse Engineering (WCRE’09)*, IEEE, pp. 303-306, 2009.

[63] O. Gordieiev et al., “Evolution of software quality models in context of the standard ISO 25010”, *In Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX* , Poland, Springer, Cham, pp. 223-232, 2014.

[64] L. B. Ammar, A. Trabelsi and A. Mahfoudhi, “A model-driven approach for usability engineering of interactive systems “, *Software Quality Journal*, vol.: 24, issue: 2, pp. 301-335, 2016.

[65] S. Winter, S. Wagner and F. Deissenboeck, “A comprehensive model of usability”, *In Proceedings of International Conference on Engineering for Human-Computer Interaction (IFIP)*, Springer, Berlin, Heidelberg, pp. 106-122, 2007.

[66] E. Shawgi, N. A. Noureldien, “Usability measurement model (umm): a new model for measuring websites usability”, *International Journal of Information Science*, vol.: 5, issue: 1, pp. 5-13, 2015.

[67] A. Seffah, E. Metzker, “The obstacles and myths of usability and software engineering”, *Communications of the ACM*, vol.: 47, issue: 12, pp. 71-76, 2004.

[68] R. G. Dromey, “A model for software product quality”, *IEEE Transactions on software engineering*, vol.: 21, issue: 2, pp. 146-162, 1995.

[69] P. N. Rao, G. V. Ramararaju, “Strategic Information Planning: Alignment of IT Planning with Business Planning with the Application of Analytic Network and Analytic Hierarchy Process Maturity model”, *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, vol.: 7, issue: 6, 2017

[70] T.L. Saaty, “Fundamentals of the Analytic Network Process”, *In Proceedings of the 5th international symposium on the analytic hierarchy process (ISAHP)*, Kobe, Japan, pp. 12-14, 1999.

[71] B. P. Lientz,, E. B. Swanson and G. E. Tompkins, “Characteristics of Application Software Maintenance”, *Communications of the ACM* , vol.: 21, issue: 6, pp. 466-471, 1978.

[72] L. Duboc , D. Rosenblum and T. Wicks, “A framework for characterization and analysis of software system scalability”, *In Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 375-384, 2007.

[73] T.H. Sheakh, S.M.K.Quadri and V. Singh, "A Critical Review of Software Reliability”, *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, vol.: 2, issue: 4, pp. 496-499, 2012.

[74] R. Lal and N. Kumar, “Design and Analysis of Reliability for Component-Based Software System by using Soft Computing Approaches”, *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, vol.: 4, issue: 6, pp. 929-932, 2014.

[75] K. Goffin, “Design for supportability: essential component of new product development”, *Research-Technology Management*, vol.: 43, issue: 2, pp. 40-47, 2000.

[76] K. Goffin, “Customer support: a cross-industry study of distribution channels and strategies”, *International Journal of Physical Distribution & Logistics Management*, vol.: 29, issue: 6, pp. 374-398, 1999.

[77] A. Yıldızbaşı and B. D. Rouyendegh, “Multi-criteria decision making approach for evaluation of the performance of computer programming languages in higher education”, *Computer Applications in Engineering Education*, vol.:26, issue: 6, pp. 1992-2001, 2018.

[78] B. Chatters, “Software Reliability Improvement-The Fault Free Factory(A Case Study)”, *Software Engineering for Large Software Systems*, Springer, Dordrecht, pp. 220-235, 1990.

[79] B. Malcolm, “A Large Embedded System Project Case Study”, *Software Engineering for Large Software Systems*, Springer, Dordrecht, pp. 96-121, 1990.

[80] A. Gupta and P. Dashore, “An Approach to Analyse Software Reusability of Object Oriented Code”, *International Journal of Research in Science & Engineering*, vol.: 3, issue: 1, 2017.

[81] B. M. Goel and P. K. Bhatia, “Analysis of Reusability of Object-Oriented System using CK Metrics”, *International Journal of Computer Applications*, vol.: 60, issue: 10, pp. 32-36, 2012.

[82] K. Dominguez et al., “Software quality model based on software development approaches”, *Software Engineering and applications*, pp. 1-6, 2007.

[83] R. Martin, “OO design quality metrics”, *An Analysis of Dependencies*, 1994.

[84] H. Yang, “Measuring software product quality with ISO standards base on fuzzy logic technique”, *Affective Computing and Intelligent Interaction, Advances in Inteeelllllligent and Soft Computing*, Springer, Berlin, vol.: 137, pp. 59-67, 2012.

[85] K. K. Yuen and H. C. Lau, “A fuzzy group analytical hierarchy process approach for software quality assurance management: Fuzzy logarithmic least squares method”, *Expert Systems with Applications*, vol.: 38, issue: 8, pp 10292-10302, 2011.

[86] A. K. Pandey and C. P. Agrawal, “Fuzzy ANP model to measure the maintainability of desktop software based on software development factors”, *Indian Journal of Science and Technology*, vol.: 9, issue: 33, pp. 1-9, 2016. DOI: 10.17485/ijst/2016/v9i33/100218

[87] U. Erdemir and F. Buzluca, “A learning-based module extraction method for object-oriented systems”, *Journal of Systems and Software*, vol.: 97, issue: 3, pp. 156-177, 2014. DOI: 10.1016/j.jss.2014.07.038

[88] A. Przybyłek, “Systems evolution and software reuse in object-oriented programming and aspect-oriented programming”, *Objects, Models,*

Components, Patterns. TOOLS 2011. Lecture Notes in Computer Science, Springer, Berlin, vol.: 6705, pp. 163-178, 2011.

[89] A. Sheshasaayee and R. Jose, "A Theoretical Framework for the Maintainability Model of Aspect Oriented Systems", *Procedia Computer Science*, vol. 62, pp. 505-512, 2015. DOI: 10.1016/j.procs.2015.08.523

[90] A. Sheshasaayee, R. Jose," A Fuzzy Approach for the Maintainability Assessment of Aspect Oriented Systems", *Information Systems Design and Intelligent Applications. Advances in Intelligent Systems and Computing*, Springer, vol.: 435, pp. 509-517, 2016. DOI: 10.1007/978-81-322-2757-1_50

[91] P. K. Singh, O. Sangwan, A. Singh and A. Pratap, "A Framework for Assessing the Software Reusability using Fuzzy Logic Approach for Aspect Oriented Software", *International Journal of Information Technology and Computer Science*, vol.: 7, issue: 2, pp. 12-20, 2015.
DOI: 10.5815/ijitcs.2015.02.02

[92] P.J. Kaur et al., "A framework for assessing reusability using package cohesion measure in aspect oriented systems", *International Journal of Parallel Program*, vol.: 46, issue: 3, pp. 543-564, 2018.
DOI: 10.1007/s10766-017-0501-6

[93] P. J. Kaur and S. Kaushal, "Package level metrics for reusability in AOS", *In proceedings of International conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*, IEEE, pp. 364-368, 2015.

[94] G. Blaschek et al., "Method and device for automatically evaluating the quality of a software source code", U.S. Patent Application 11/991,429, filed February 26, 2009.

[95] Y. Nir-Buchbinder et al., "Cross-concern code coverage assessment", U.S. Patent 8,607,198, issued December 10, 2013.

[96] S. Sarkar et al., "*Measuring quality of software modularization*", U.S. Patent 8,146,058, issued March 27, 2012.

[97] M. I. Ghareb, G.Allen, “State of the art metrics for aspect oriented programming”, *In Proceedings of AIP conference*, Melville, vol.: 1952, issue: 1, 2018. DOI: 10.1063/1.5032069

[98] E. Kirubakaran and K. R. Martin KR,”A review on coupling metrics in aspect oriented system”, *International Journal of Control Theory and Applications*, vol.: 9, pp. 93-97, 2016.

[99] K. Mik, “Aop@ work: aop tools comparison”, IBM Developer Works, India, 2005

[100] M. Sandip, K. Rajnish and S. Kumar, “Package level cohesion metric for object-oriented design”, *International Journal of Engineering Technology (IJET)*, vol.: 5, pp. 2523-2528, 2013.

[101] L. C. Briand, S. Morasca and V. R. Basili, “Property-based software engineering measurement”, *In IEEE Transactions on Software Engineering*, vol.: 22, issue: 1, pp. 68-86, 1996. DOI: 10.1109/32.481535

[102] K.P. Srinivasan and T. Devi, “Software metrics validation methodologies in software engineering”, *International Journal of Software Engineering & Applications (IJSEA)*, vol.: 5, issue: 6, pp. 87-102, 2014.

[103] J. I. Panach et al., “Early usability measurement in model-driven development: Definition and empirical evaluation”, *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol.: 21, issue: 3, pp. 339-365, 2011. DOI: 10.1142/S0218194011005311

[104] A. Sharma, R. Kumar and P. S. Grover, “Empirical evaluation and validation of interface complexity metrics for software components”, *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol.: 18, issue: 7, pp. 919-931, 2008.
DOI: 10.1142/S0218194008003957

[105] M. Bertoa and A. Vallecillo, “Usability metrics for software components”, *In Proceedings of 8th international workshop on quantitative approaches in object-oriented software engineering (QAOOSE’2004)*, Oslo, Norway 2004.

[106] K. Moumane, A. Idri and A. Abran, “Usability evaluation of mobile applications using ISO 9241 and ISO 25062 standards”, SpringerPlus, vol.: 5, issue: 548, 2016. DOI: 10.1186/s40064-016-2171-z

[107] K. Z. Winn, “Quantifying and Validation of Changeability and Extensibility for Aspect-Oriented Software”, *In Proceedings of International Conference of Advances in Engineering and Technology*, pp. 162-166, 2014.

[108] A. Kaur, P.S. Grover and A. Dixit, “Quantitative evaluation of proposed maintainability model using AHP method”, *In IEEE Proceedings of 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE (Proceedings in SCOPUS), pp. 1367-1371, 2015.

[109] A. Kaur, P.S. Grover and A. Dixit, “Analysis of Reliability Model with the application of MCDM”, *International Journal of Emerging Technology and Advanced Engineering (IJETAE),* ISO 9001:2008 Certified Journal, vol.: 8, issue: 3, pp. 250-255, ISSN 2250-2459, 2018.

[110] Rafa E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and Comparative Study”, *Journal of American Science*, vol.: 6, issue: 3, pp. 166-175, 2010.

[111] AspectJ project, http://eclipse.org/aspectj/

[112] H. P. Breivold and I. Crnkovic, “Analysis of Software Evolvability in Quality Models," *In Proceedings of 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2009),* pp. 279-282, 2009. DOI: 10.1109/SEAA.2009.10

[113] “ISO/IEC/IEEE International Standard - Systems and software engineering -- Vocabulary", ISO/IEC/IEEE 24765:2010(E*)*, pp 1-418, 2011. DOI: 10.1109/IEEESTD.2010.5733835

[114] “ISO/IEC/IEEE International Standard - Systems and software engineering--Vocabulary," ISO/IEC/IEEE 24765:2017(E), pp.1-541, 2017. DOI: 10.1109/IEEESTD.2017.8016712

[115] F. B. Abreu, R. Esteves and M. Goulao, "The design of eiffel programs: Quantitative evaluation using the mood metrics", *In Proceedings of TOOLS'96*, 1996.

[116] K. Wiegers and J. Beatty, "Software requirements", Pearson Education, 2013.

[117] P. J. Kaur, and S. Kaushal, "Cohesion and coupling measures for aspect oriented systems", Elsevier, AETS/7/590, vol.: 7, pp. 784-788, 2013.

[118] P.C. Fishburn, "Letter to the editor—additive utilities with incomplete product sets: application to priorities and assignments", *Operations Research*, vol.: 15, issue: 3, pp. 537-542, 1967.

[119] E. Triantaphyllou, "Multi-Criteria Decision Making: A Comparative Study", Kluwer Academic Publishers (now Springer), Dordrecht, The Netherlands, pp. 320. ISBN 0-7923-6607-7.

[120] R. Attri, N. Dev and V. Sharma, "Interpretive structural modelling (ISM) approach: an overview", *Research Journal of Management Sciences*, vol.: 2, issue: 2, pp. 3-8, 2013.

[121] A. Assari, T. Mahesh and E. Assari,"Role of public participation in sustainability of historical city: usage of TOPSIS method", *Indian Journal of Science and Technology*, vol.: 5, issue: 3, pp. 2289-2294, 2012.

[122] Metrics 1.3.6 Tool http://metrics.sourceforge.net/update

APPENDIX –A

In order to evaluate the consistency to the proposed characteristics in the respective models through Analytical Hierarchical Process, an input form is designed to collects the input from the personnel of the software industry. Four forms are created for the collection of input for four proposed characteristics namely

I. Extensibility Characteristic in Software Maintainability Model
II. Optimized code Characteristic in Performance Efficiency Model
III. Scalability Characteristic in Software Reliability Model
IV. Supportability Characteristic in Software Usability Model

The screenshot of the form prepared for Extensibility Characteristic in Software Maintainability Model is as follows:

NAME:

DOMAIN:

Scale: 1 - Equal Importance, 3 - Moderate importance, 5 - Strong importance, 7 - Very strong importance, 9 - Extreme importance (2,4,6,8 values in-between).

There are 15 pairwise comparisons. Please specify the following:-

Which criterion is more important, and how much more on a scale 1 to 9?

	A - Importance - or B?			Equal	How much more?
1	extensibility	or	reusability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
2	extensibility	or	modifiability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
3	extensibility	or	analysability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
4	extensibility	or	testability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
5	extensibility	or	modularity	**1**	2 **3** 4 **5** 6 **7** 8 **9**

Figure A1.1: Input Form for Maintainability Model

6	reusability	or modifiability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
7	reusability	or analysability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
8	reusability	or testability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
9	reusability	or modularity	**1**	2 **3** 4 **5** 6 **7** 8 **9**
10	modifiability	or analysability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
11	modifiability	or testability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
12	modifiability	or modularity	**1**	2 **3** 4 **5** 6 **7** 8 **9**
13	analysability	or testability	**1**	2 **3** 4 **5** 6 **7** 8 **9**
14	analysability	or modularity	**1**	2 **3** 4 **5** 6 **7** 8 **9**
15	testability	or modularity	**1**	2 **3** 4 **5** 6 **7** 8 **9**

Figure A1.2: Input Form for Maintainability Model (Cont...)

The terms and definitions are also provided with the input form specifying the meanings of the terms and definations used in the data collection for for maintainability model.

1. Maintainability:
    a. Extensibility : the degree to extend or enhance the current system and the level of effort required to implement the extension, while minimizing impact to existing system functions.
    b. Reusability : degree to which an asset can be used in more than one system, or in building other assets.
    c. Modifiability: degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
    d. Analyzability : degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
    e. Testability : degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.
    f. Modularity :degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components

Figure A1.3: Terms and Definitions for Maintainability Model

The screenshot of the form prepared for Scalability Characteristic in Reliability Model is as follows:

Name:

Domain:

Scale: 1 - Equal Importance, 3 - Moderate importance, 5 - Strong importance, 7 - Very strong importance, 9 - Extreme importance (2,4,6,8 values in-between).

There are 10 pairwise comparisons in relation to software reliability. Please specify the following:-

Which criterion is more important, and how much more on a scale 1 to 9?

	A - Importance - or B?			Equal	How much more?
1	scalability	or	maturity	1	2 3 4 5 6 7 8 9
2	scalability	or	availability	1	2 3 4 5 6 7 8 9
3	scalability	or	fault tolerance	1	2 3 4 5 6 7 8 9
4	scalability	or	recoverability	1	2 3 4 5 6 7 8 9
5	maturity	or	availability	1	2 3 4 5 6 7 8 9
6	maturity	or	fault tolerance	1	2 3 4 5 6 7 8 9
7	maturity	or	recoverability	1	2 3 4 5 6 7 8 9
8	availability	or	fault tolerance	1	2 3 4 5 6 7 8 9
9	availability	or	recoverability	1	2 3 4 5 6 7 8 9
10	fault tolerance	or	recoverability	1	2 3 4 5 6 7 8 9

Figure A1.4: Input Form for Reliability Model

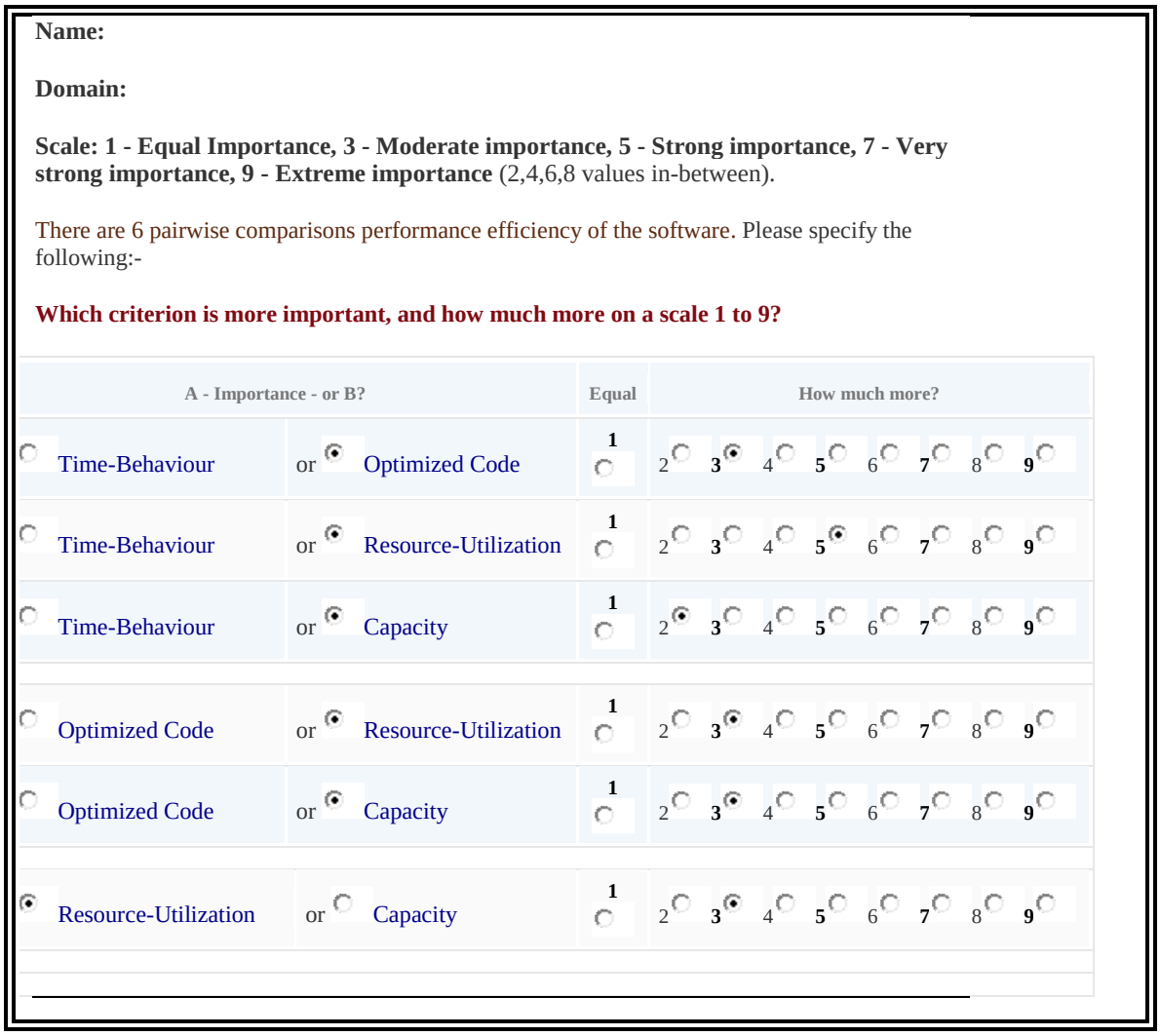
The terms and definitions are also provided with the input form specifying the meanings of the terms and definitions used in the data collection for Reliability model.

Terms and definitions

2) **Reliability -** degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.
    a. **Maturity -** degree to which a system, product or component meets needs for reliability under normal operation.
    b. **Availability -** degree to which a system, product or component is operational and accessible when required for use.
    c. **Fault Tolerance -** degree to which a system, product or component operates as intended despite the presence of hardware or software faults
    d. **Recoverability -** degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.
    e. **Scalability -** ability of the system to either handle increase in load without impact on the performance of the system or the ability to be readily enlarged.

Figure A1.5: Terms and Definitions for Reliability Model

The screenshot of the form prepared for Optimized Code Characteristic in Performance Efficiency Model is as follows:



Name:

Domain:

Scale: 1 - Equal Importance, 3 - Moderate importance, 5 - Strong importance, 7 - Very strong importance, 9 - Extreme importance (2,4,6,8 values in-between).

There are 6 pairwise comparisons performance efficiency of the software. Please specify the following:-

Which criterion is more important, and how much more on a scale 1 to 9?

A - Importance - or B?			Equal	How much more?
○	Time-Behaviour	or ◉ Optimized Code	1 ○	2 ○ **3** ◉ 4 ○ **5** ○ 6 ○ **7** ○ 8 ○ **9** ○
○	Time-Behaviour	or ◉ Resource-Utilization	1 ○	2 ○ **3** ○ 4 ○ **5** ◉ 6 ○ **7** ○ 8 ○ **9** ○
○	Time-Behaviour	or ◉ Capacity	1 ○	2 ◉ **3** ○ 4 ○ **5** ○ 6 ○ **7** ○ 8 ○ **9** ○
○	Optimized Code	or ◉ Resource-Utilization	1 ○	2 ○ **3** ◉ 4 ○ **5** ○ 6 ○ **7** ○ 8 ○ **9** ○
○	Optimized Code	or ◉ Capacity	1 ○	2 ○ **3** ◉ 4 ○ **5** ○ 6 ○ **7** ○ 8 ○ **9** ○
◉	Resource-Utilization	or ○ Capacity	1 ○	2 ○ **3** ◉ 4 ○ **5** ○ 6 ○ **7** ○ 8 ○ **9** ○

Figure A1.6: Input Form for Performance Efficiency Model

The screenshot of the terms and definitions are also provided with the input form specifying the meanings of the terms and definitions used in the data collection for performance efficiency model.

Terms and definitions

3) **Performance Efficiency** - Performance relative to the amount of resources used under stated conditions. Resources can include other software products, the software and hardware configuration of the system, and materials (e.g. print paper, storage media).

   a. **Time Behavior -** degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements
   b. **Resource Utilization -** degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements
   c. **Capacity -** degree to which the maximum limits of a product or system parameter meet requirements . Parameters can include the number of items that can be stored, the number of concurrent users, the communication bandwidth, throughput of transactions, and size of database.
   d. **Optimized Code -** Code written by consistently applying well coding standard and proper coding techniques **.**

Figure A1.7: Terms and Definitions for Performance Efficiency Model

The screenshot of the form prepared for Supportability Characteristic in Usability Model is as follows:

Name: Domain -Storage

Scale: 1 - Equal Importance, 3 - Moderate importance, 5 - Strong importance, 7 - Very strong importance, 9 - Extreme importance (2,4,6,8 values in-between).

There are 15 pairwise comparisons in relation to software usability. Please specify the following:-

Which criterion is more important, and how much more on a scale 1 to 9?

	A - Importance - or B?		Equal	How much more?
1	Appropriateness Recognisability	or Supportability	1	2 **3** 4 **5** 6 **7** 8 **9**
2	Appropriateness Recognisability	or Learnability	1	2 **3** 4 **5** 6 **7** 8 **9**
3	Appropriateness Recognisability	or Operability	1	2 **3** 4 **5** 6 **7** 8 **9**
4	Appropriateness Recognisability	or User error protection	1	2 **3** 4 **5** 6 **7** 8 **9**
5	Appropriateness Recognisability	or User interface aesthetics	1	2 **3** 4 **5** 6 **7** 8 **9**
6	Appropriateness Recognisability	or Accessibility	1	2 **3** 4 **5** 6 **7** 8 **9**

Figure A1.8: Input Form for Usability Model

7	Supportability	or Learnability	1	2 3 4 5 6 7 8 9
8	Supportability	or Operability	1	2 3 4 5 6 7 8 9
9	Supportability	or User error protection	1	2 3 4 5 6 7 8 9
10	Supportability	or User interface aesthetics	1	2 3 4 5 6 7 8 9
11	Supportability	or Accessibility	1	2 3 4 5 6 7 8 9
12	Learnability	or Operability	1	2 3 4 5 6 7 8 9
13	Learnability	or User error protection	1	2 3 4 5 6 7 8 9
14	Learnability	or User interface aesthetics	1	2 3 4 5 6 7 8 9
15	Learnability	or Accessibility	1	2 3 4 5 6 7 8 9
16	Operability	or User error protection	1	2 3 4 5 6 7 8 9
17	Operability	or User interface aesthetics	1	2 3 4 5 6 7 8 9
18	Operability	or Accessibility	1	2 3 4 5 6 7 8 9
19	User error protection	or User interface aesthetics	1	2 3 4 5 6 7 8 9

Figure A1.9: Input Form for Usability Model (Cont...)

The terms and definitions are also provided with the input form specifying the meanings of the terms and definitions used in the data collection for usability model.

Terms and definitions

4) **Usability -** degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use

   a. **Appropriateness Recognizability -** degree to which users can recognize whether a product or system is appropriate for their needs. Appropriateness recognizability will depend on the ability to recognize the appropriateness of the product or system's functions from initial impressions of the product or system and/or any associated documentation.
   b. **Learnability-** degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.
   c. **Operability-** degree to which a product or system has attributes that make it easy to operate and control.
   d. **User Error Protection-** degree to which a system protects users against making errors.
   e. **User Interface Aesthetics -** degree to which a user interface enables pleasing and satisfying interaction for the user.
   f. **Accessibility-** degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.
   g. **Supportability** - ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.

Figure A1.10: Terms and Definitions for Usability Model

APPENDIX –B

Statistics collected for Spacewar game developed in Java programming language using Plug-In Metrics 1.3.6 tool on the Eclipse Platform. The screen shot of the collected statistics is shown as follows.

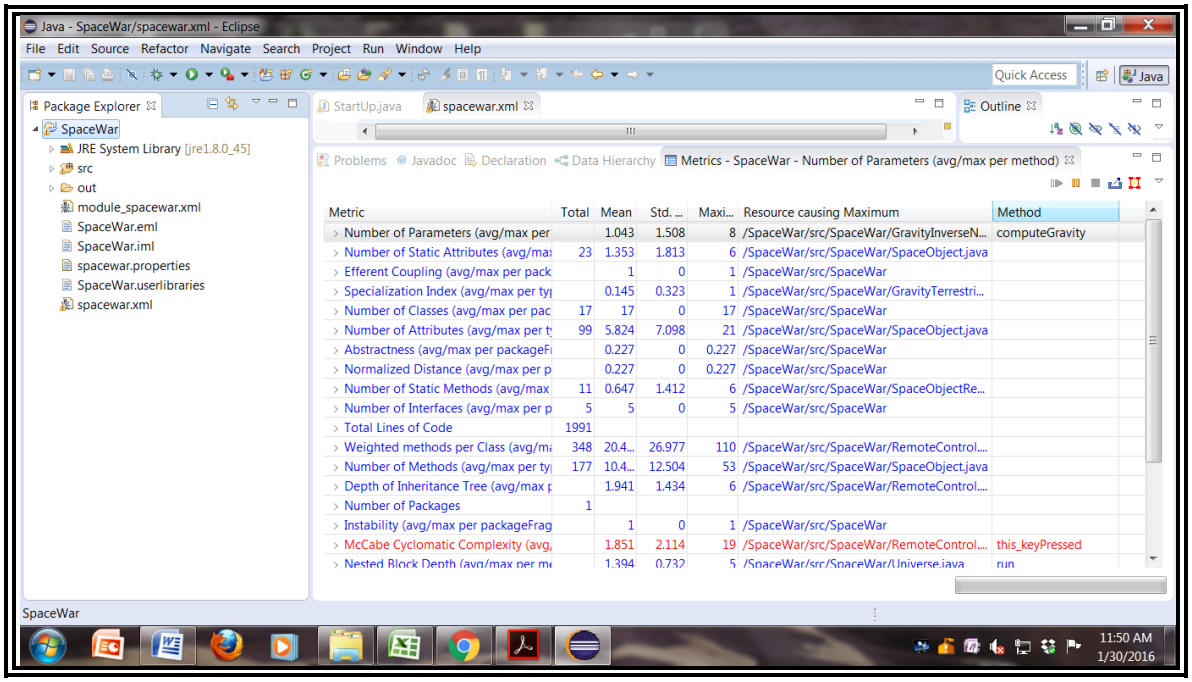


Figure A2.1: Statistics collected for Spacewar Java.

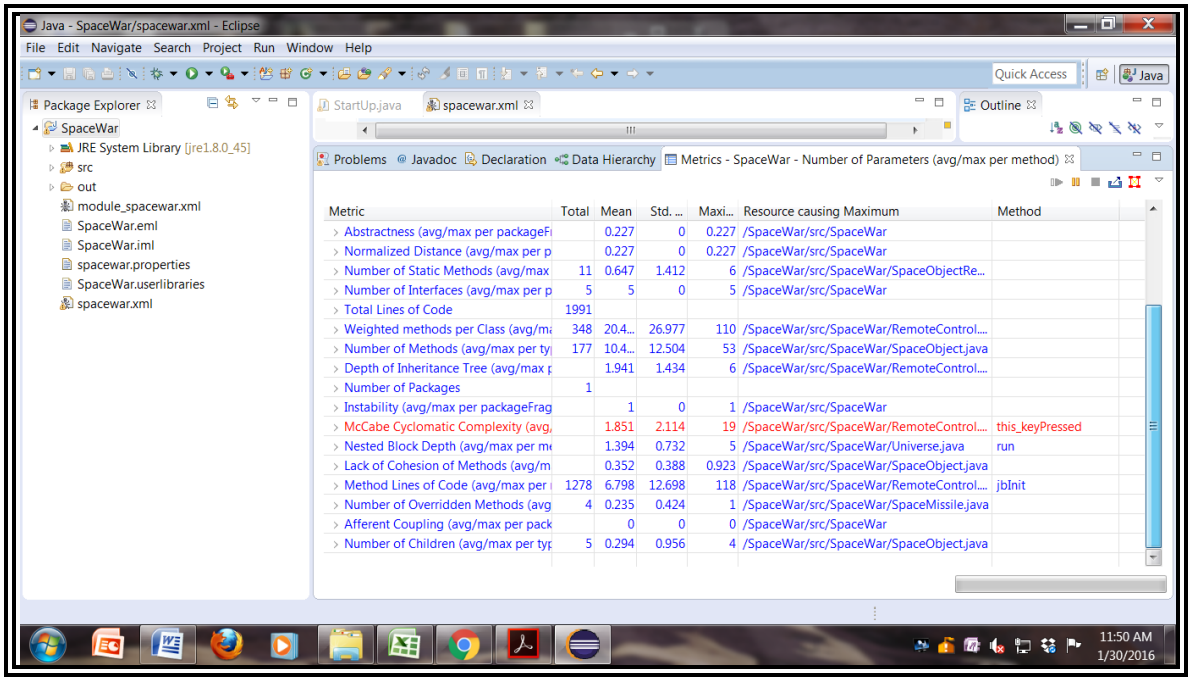


Figure A2.2: Statistics collected for Spacewar Java (Cont...)

Statistics collected for Spacewar game developed in AspectJ programming language using the same Plug In on the Eclipse Platform as Java version to maintain the consistency. The screen shot of the collected statistics is shown as follows.

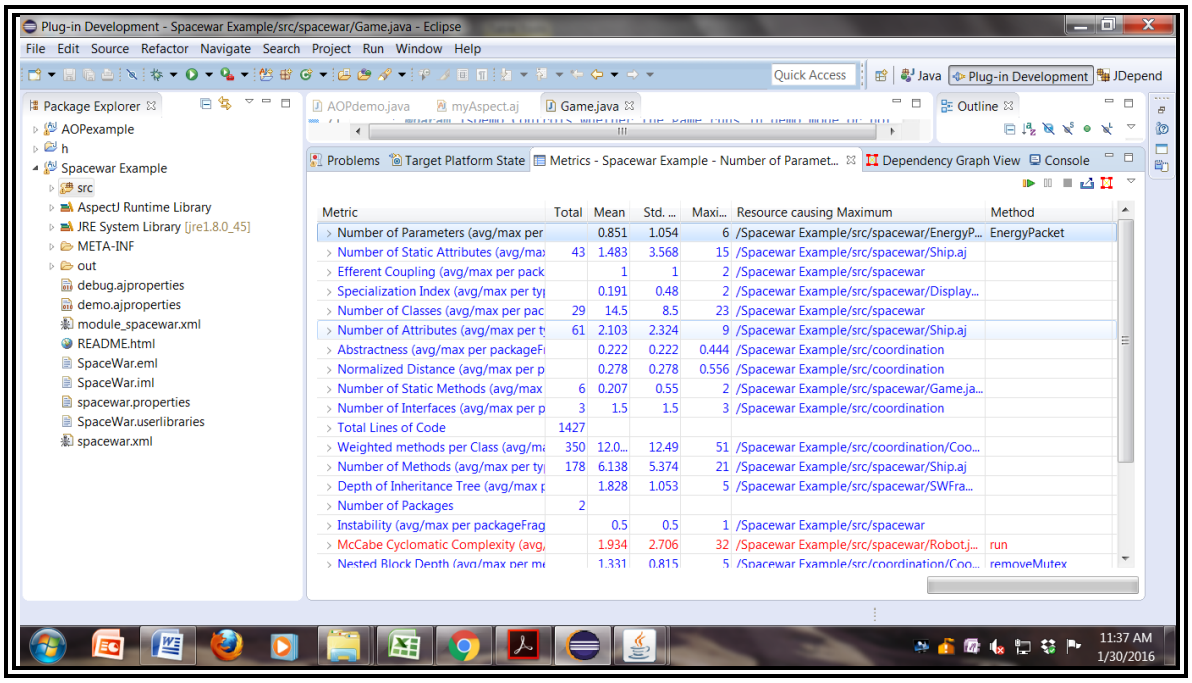


Figure A2.3: Statistics collected for Spacewar AspectJ.

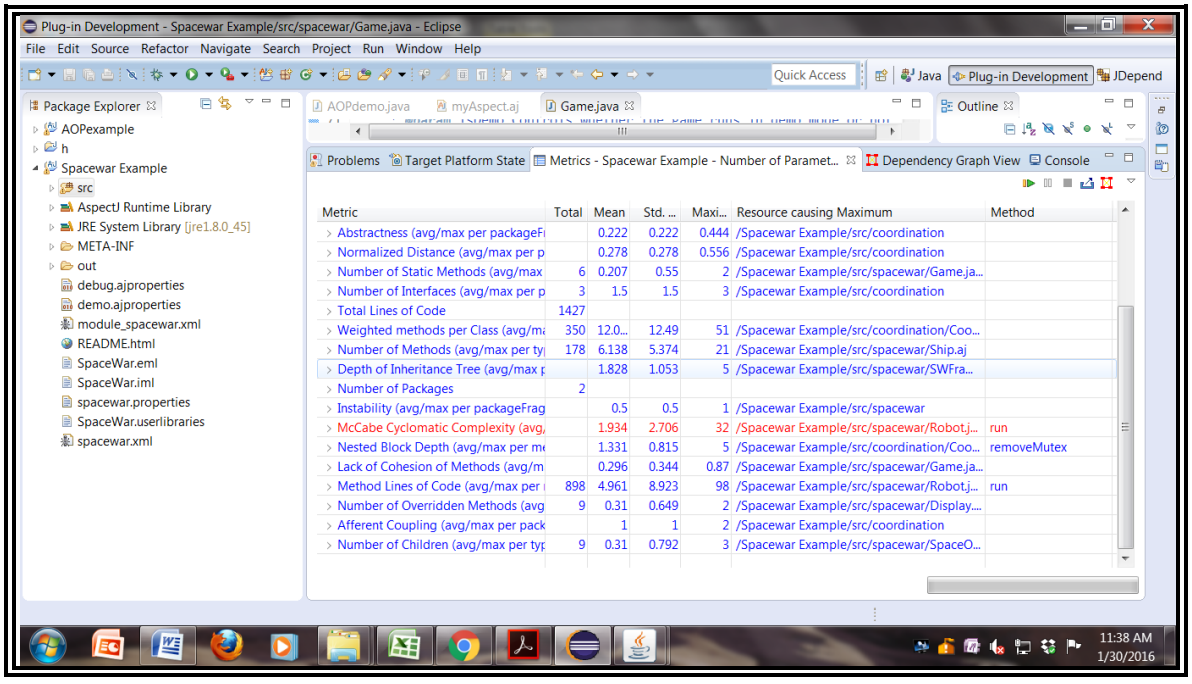


Figure A2.4: Statistics collected for Spacewar AspectJ (Cont...)

APPENDIX –C

Screenshot of the Sample Questionnaire is presented as follows.

Table A3.1: Supportability Questionnaire

S.No.	*Supportability Questionnaire*	Predominantly Disagree	Disagree	Satisfactory	Agree	Predominantly Agree
		1	2	3	4	5
1	*The user manual provided is easy to understand.*					
2	*The help menu is easy to locate at the time of work.*					
3	*The help files readily contains the context of actual usage at the time of work.*					
4	*The commands that I require at the time of work are easy to locate.*					
5	*Troubleshooting popup appears automatically when ever and where ever the system struck.*					
6	*The user manual contains all the relevant information regarding the usage of the software.*					
7	*Event logging option is available at time of problem occurrence.*					
8	*Report an Error' option popup at the time of logging error.*					
9	*Terms used in the documentation is unambiguous and clear.*					
10	*Software never breaks down abruptly without giving proper descriptive error message.*					
11	*Diagrams are used in the documentation to explain the usage of the software.*					
12	*Code tracing option is available and works correctly*					
13	*Colour scheme applied in code tracing is helpful*					
14	*Demo videos are available to show the usage of the software and on how to use help options*					
15	*Software documentation is useful and is understandable by physical handicaps.*					

The response of the supportability questionnaire as collected from the 5 evaluators for the two Projects is given as under:

Table A3.2: Supportability Questionnaire response from 5 Evaluators for Project 1

Project 1						
S.No.	*Supportability Questionnaire*	Predominantly Disagree	Disagree	Satisfactory	Agree	Predominantly Agree
		1	2	3	4	5
		u1	u2	u3	u4	u5
1	*The user manual provided is easy to understand.*	3	3	2	2	3
2	*The help menu is easy to locate at the time of work.*	2	2	1	2	1
3	*The help files readily contains the context of actual usage at the time of work.*	2	2	1	2	1
4	*The commands that I require at the time of work are easy to locate.*	4	3	2	2	2
5	*Troubleshooting popup appears automatically when ever and where ever the system struck.*	1	1	1	2	3
6	*The user manual contains all the relevant information regarding the usage of the software.*	3	2	2	2	3
7	*Event logging option is available at time of problem occurrence.*	1	1	1	2	1
8	*Report an Error' option popup at the time of logging error.*	1	1	1	2	1
9	*Terms used in the documentation is unambiguous and clear.*	3	2	2	2	1
10	*Software never breaks down abruptly without giving proper descriptive error message.*	2	2	1	2	3
11	*Diagrams are used in the documentation to explain the usage of the software.*	4	3	2	2	2
12	*Code tracing option is available and works correctly*	2	2	1	2	1
13	*Colour scheme applied in code tracing is helpful*	4	3	2	2	3
14	*Demo videos are available to show the usage of the software and on how to use help options*	1	1	1	2	1
15	*Software documentation is useful and is understandable by physical handicaps.*	1	1	1	2	1

Table A3.3: Supportability Questionnaire response from 5 Evaluators for Project 2

Project 2						
S.No.	*Supportability Questionnaire*	Predominantly Disagree	Disagree	Satisfactory	Agree	Predominantly Agree
		1	2	3	4	5
		u1	u2	u3	u4	u5
1	*The user manual provided is easy to understand.*	4	3	2	2	3
2	*The help menu is easy to locate at the time of work.*	4	4	4	4	4
3	*The help files readily contains the context of actual usage at the time of work.*	4	4	3	3	2
4	*The commands that I require at the time of work are easy to locate.*	4	4	3	4	4
5	*Troubleshooting popup appears automatically when ever and where ever the system struck.*	3	3	3	2	3
6	*The user manual contains all the relevant information regarding the usage of the software.*	5	4	4	3	4
7	*Event logging option is available at time of problem occurrence.*	2	2	2	2	2
8	*Report an Error' option popup at the time of logging error.*	2	1	1	2	2
9	*Terms used in the documentation is unambiguous and clear.*	4	4	4	4	4
10	*Software never breaks down abruptly without giving proper descriptive error message.*	3	3	3	3	3
11	*Diagrams are used in the documentation to explain the usage of the software.*	5	5	4	4	5
12	*Code tracing option is available and works correctly*	2	2	2	2	2
13	*Colour scheme applied in code tracing is helpful*	3	3	4	4	3
14	*Demo videos are available to show the usage of the software and on how to use help options*	3	3	4	3	4
15	*Software documentation is useful and is understandable by physical handicaps.*	2	2	2	2	2

APPENDIX –D

Statistics collected for the selected list of AspectJ projects.

1. Spacewar AspectJ – Given in Appendix B
2. Bean AspectJ

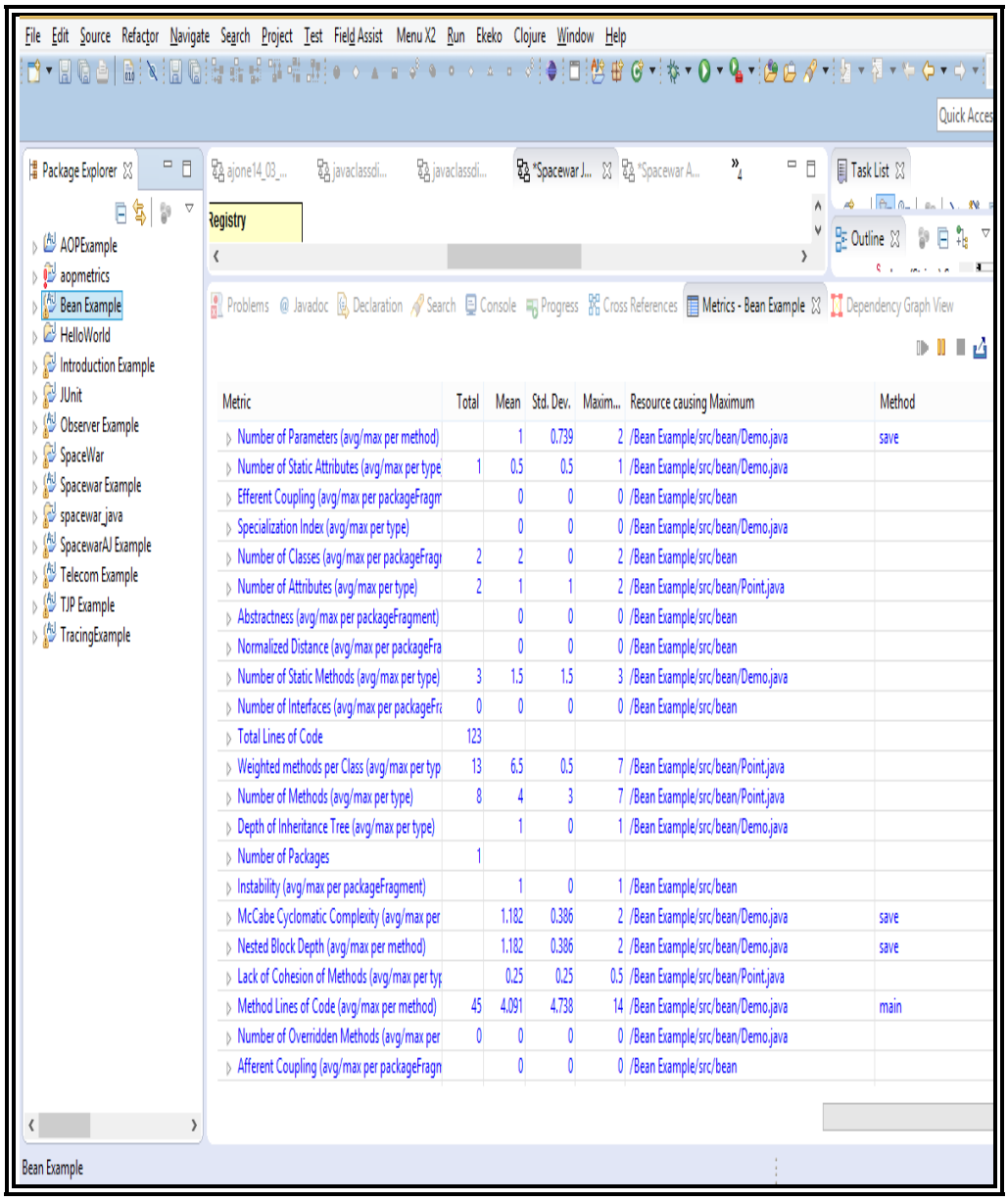


Figure A4.1: Statistics Collected for Bean AspectJ Project

3. Introduction AspectJ

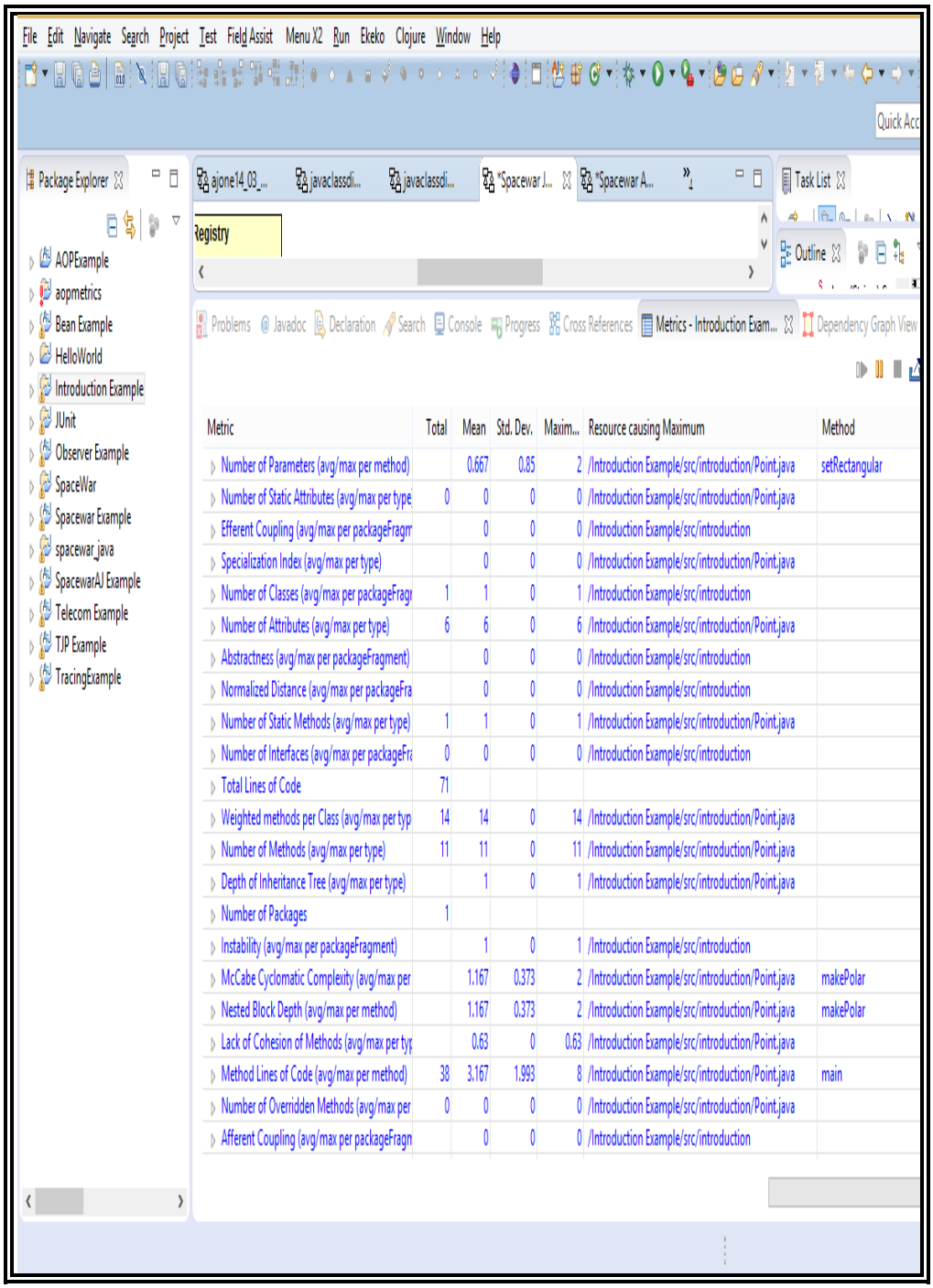
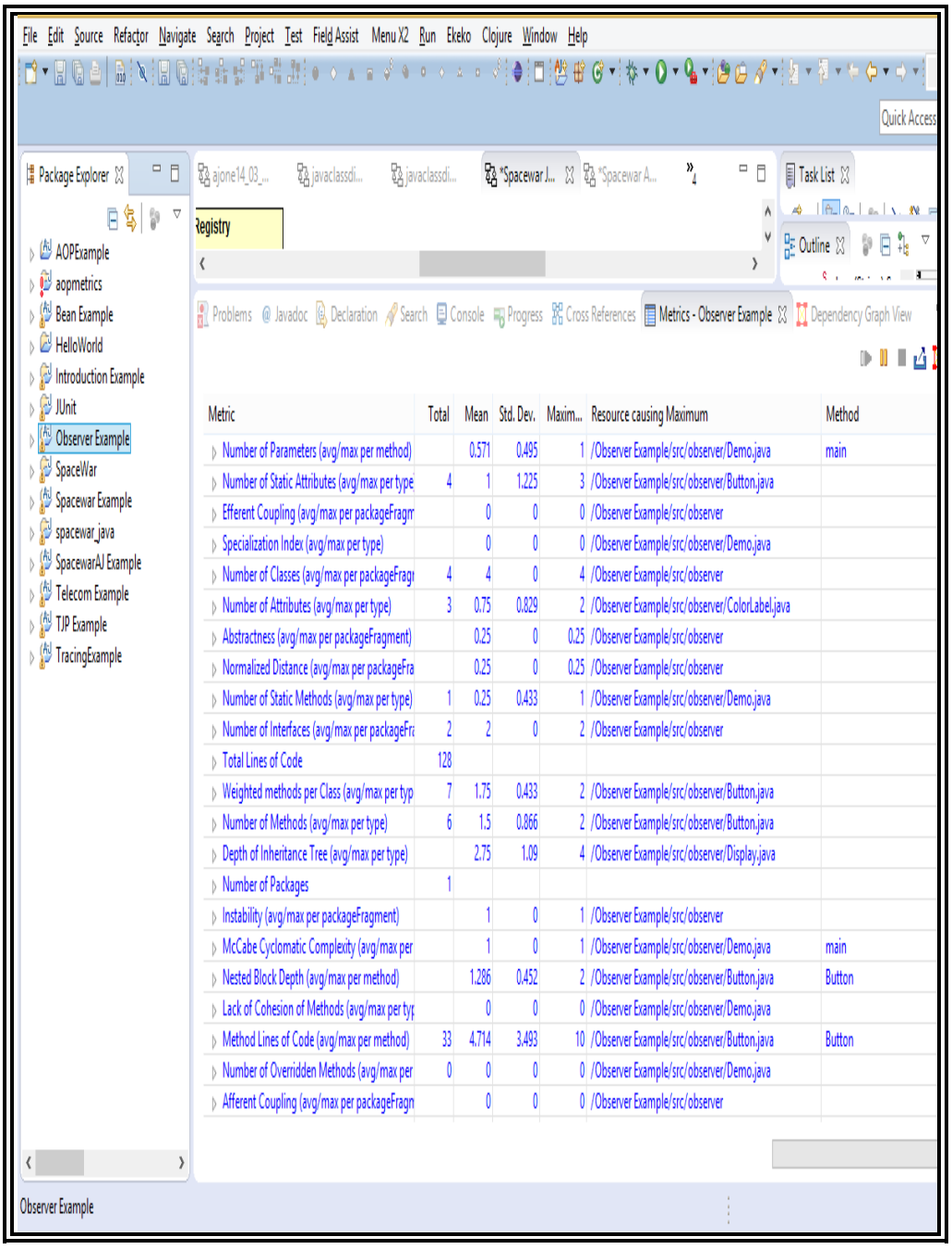


Figure A4.2: Statistics Collected for Introduction AspectJ Project

4. Observer AspectJ



Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		0.571	0.495	1	/Observer Example/src/observer/Demo.java	main
Number of Static Attributes (avg/max per type	4	1	1.225	3	/Observer Example/src/observer/Button.java	
Efferent Coupling (avg/max per packageFragm		0	0	0	/Observer Example/src/observer	
Specialization Index (avg/max per type)		0	0	0	/Observer Example/src/observer/Demo.java	
Number of Classes (avg/max per packageFragi	4	4	0	4	/Observer Example/src/observer	
Number of Attributes (avg/max per type)	3	0.75	0.829	2	/Observer Example/src/observer/ColorLabel.java	
Abstractness (avg/max per packageFragment)		0.25	0	0.25	/Observer Example/src/observer	
Normalized Distance (avg/max per packageFra		0.25	0	0.25	/Observer Example/src/observer	
Number of Static Methods (avg/max per type)	1	0.25	0.433	1	/Observer Example/src/observer/Demo.java	
Number of Interfaces (avg/max per packageFra	2	2	0	2	/Observer Example/src/observer	
Total Lines of Code	128					
Weighted methods per Class (avg/max per typ	7	1.75	0.433	2	/Observer Example/src/observer/Button.java	
Number of Methods (avg/max per type)	6	1.5	0.866	2	/Observer Example/src/observer/Button.java	
Depth of Inheritance Tree (avg/max per type)		2.75	1.09	4	/Observer Example/src/observer/Display.java	
Number of Packages	1					
Instability (avg/max per packageFragment)		1	0	1	/Observer Example/src/observer	
McCabe Cyclomatic Complexity (avg/max per		1	0	1	/Observer Example/src/observer/Demo.java	main
Nested Block Depth (avg/max per method)		1.286	0.452	2	/Observer Example/src/observer/Button.java	Button
Lack of Cohesion of Methods (avg/max per typ		0	0	0	/Observer Example/src/observer/Demo.java	
Method Lines of Code (avg/max per method)	33	4.714	3.493	10	/Observer Example/src/observer/Button.java	Button
Number of Overridden Methods (avg/max per	0	0	0	0	/Observer Example/src/observer/Demo.java	
Afferent Coupling (avg/max per packageFragn		0	0	0	/Observer Example/src/observer	

Figure A4.3: Statistics Collected for Observer AspectJ Project

5. Telecom AspectJ

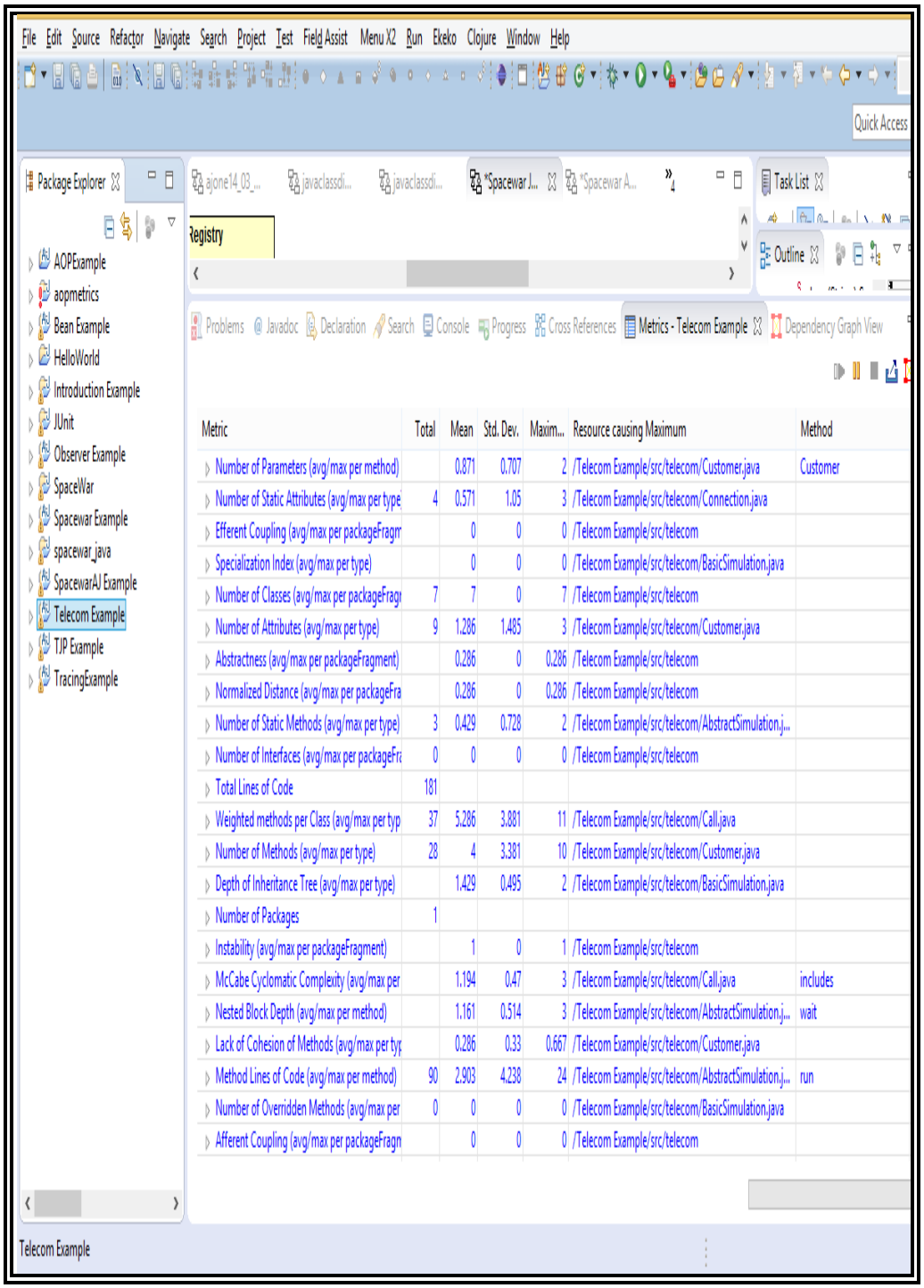


Figure A4.4: Statistics Collected for Telecom AspectJ Project

6. TJP AspectJ

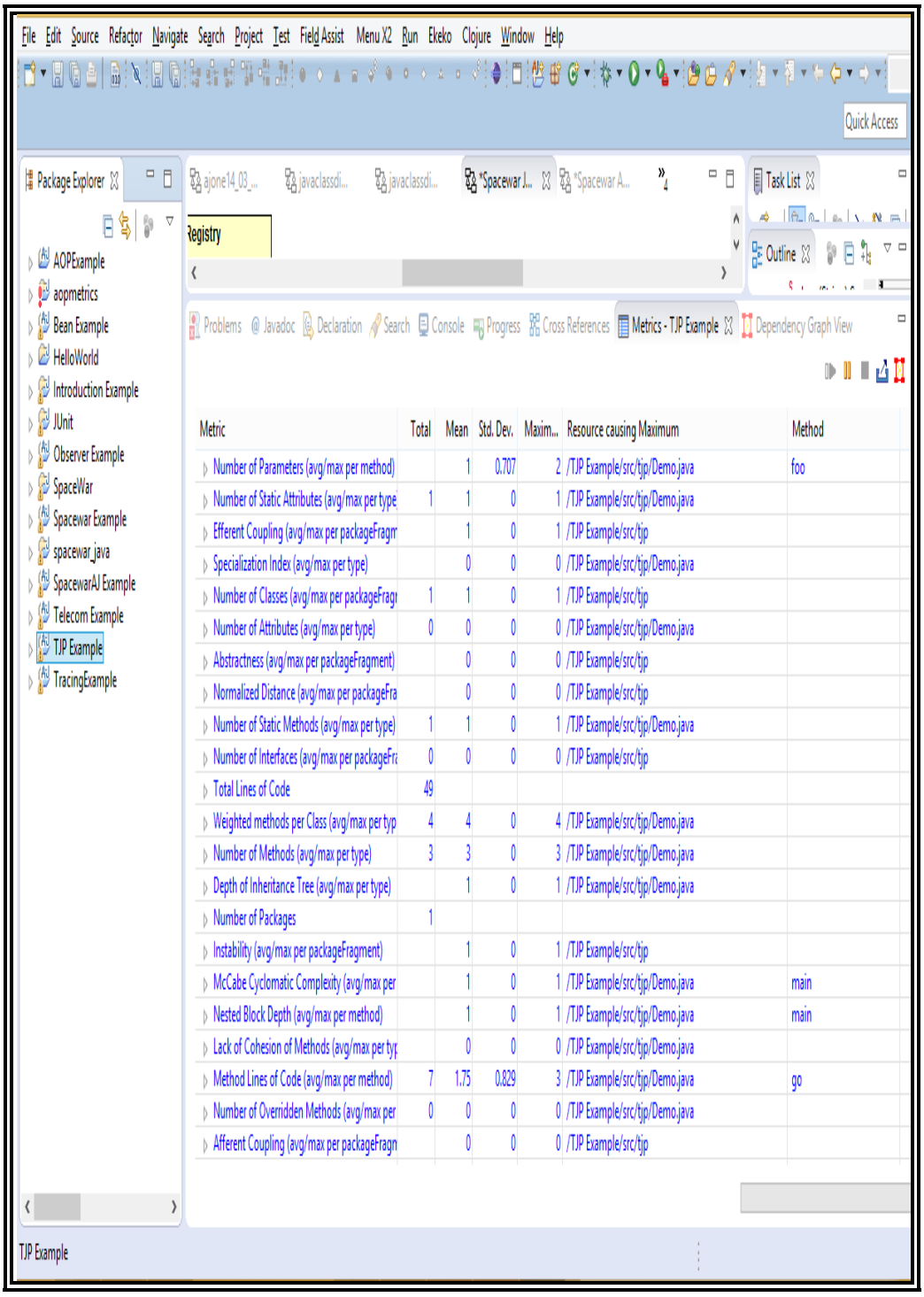


Figure A4.5: Statistics Collected for TJP AspectJ Project

7. Tracing

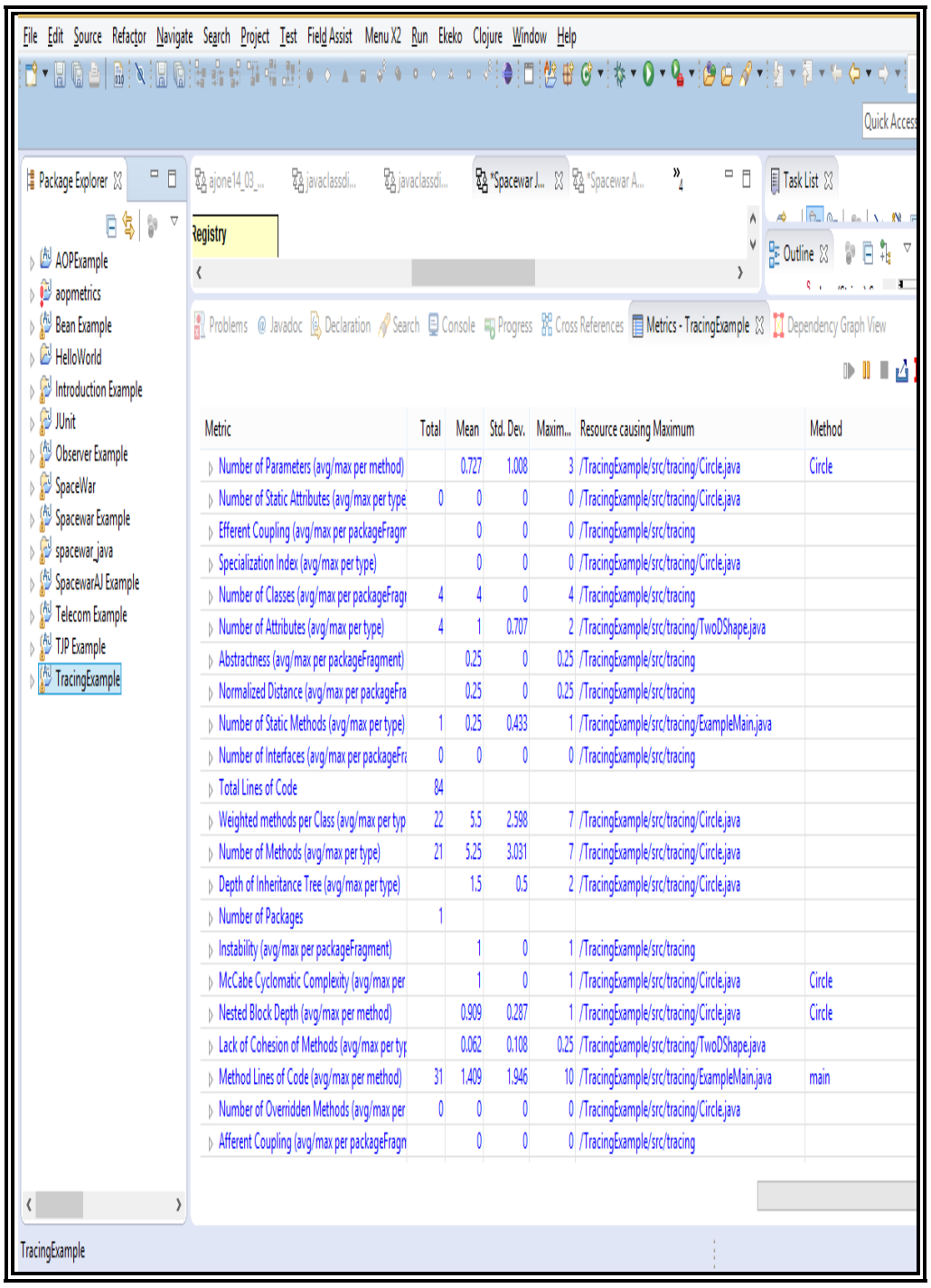


Figure A4.6: Statistics Collected for Tracing AspectJ Project

8. AJHotDraw

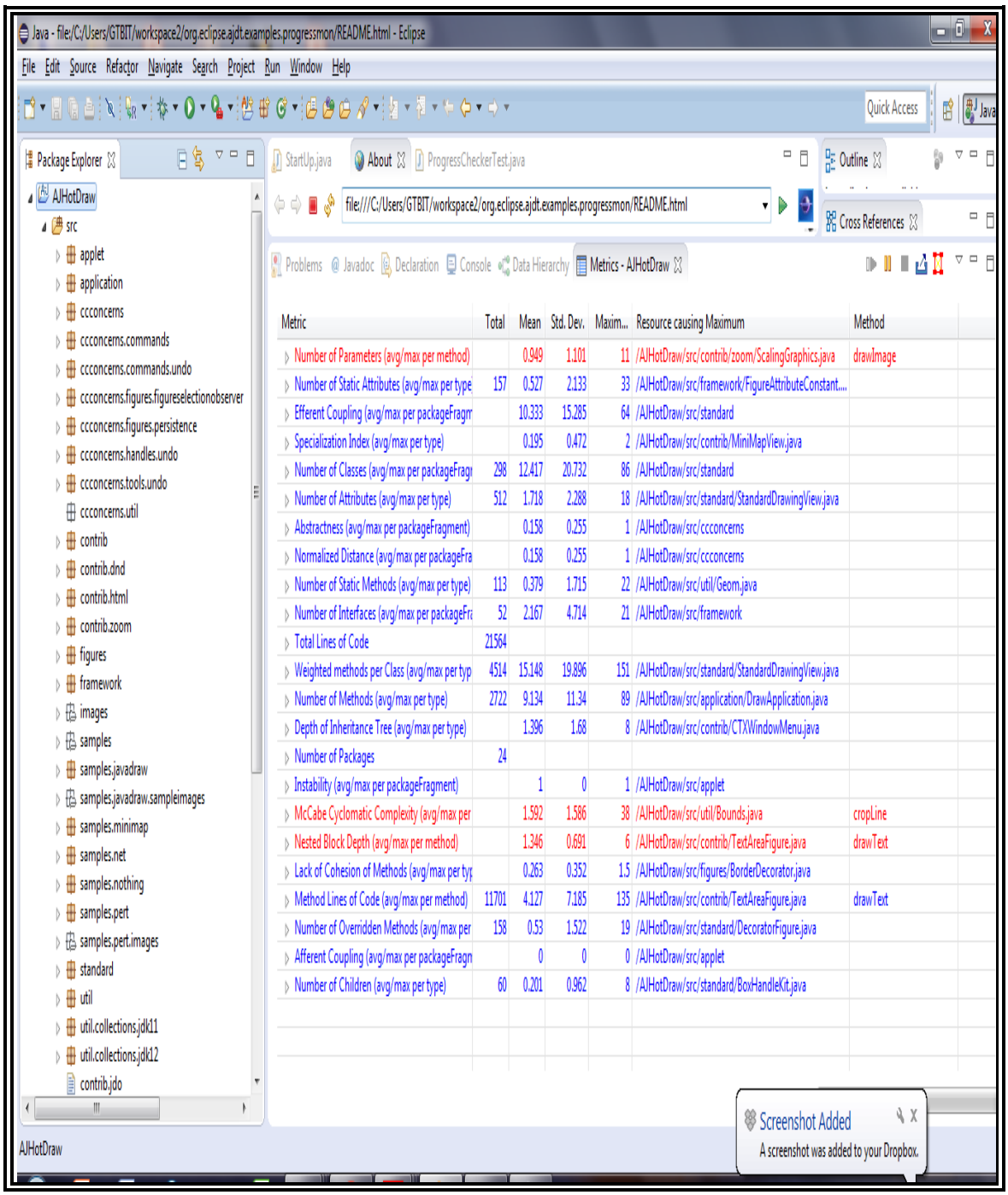


Figure A4.7: Statistics Collected for AJHotDraw AspectJ Project

BRIEF PROFILE OF RESEARCH SCHOLAR

Ms. Amandeep Kaur has received her M.Tech in Information Technology from Guru Gobind Singh Inderprastha University in the year 2008 and M.Sc. in Computer Science from Maharishi Dayanand University in the year 2004. She has more than 15 years of teaching experience. Presently she is working as Assistant Professor at Guru Tegh Bahadur Institute of Technology, Guru Gobind Singh Inderprastha University. Her research interests include Software Quality, Software Metrics, Programming Languages, Software Testing and Database Management System. She has authored research papers in various international journals and conferences.

LIST OF PUBLICATIONS

List of Published Papers : International Journals

Sr. No.	Title of the Paper along with Volume Issue No, Year of Publication	Publisher	Impact Factor	Whether Referred or Non Referred	Whether you paid any money for publication	Remarks
1	"Aspect-oriented system coupling metric and its validation", *Recent Patents on Computer Science*, vol.: 12, 2019, ISSN: 1874-4796 DOI: **10.2174/2213275912666190410143540**	Bentham Science	SJR= .17	Referred	No	**SCOPUS**
2	"A framework for evaluating extensibility in an Aspect-oriented software system and its validation", Recent Patents on Engineering, vol.: 13, 2019,ISSN: 1872-2121 DOI: **10.2174/1872121113666190625115111**	Bentham Science	SJR= 0.136	Referred	No	**SCOPUS**
3	"Critical Assessment of Aspect Orientation Metrics and Quality", Recent Trends in Programming Languages, Volume 5, Issue 2, 2018, ISSN : 2455-1821	CELNET		Referred	No	**UGC Approved**
4	"Investigation of Reusability & Complexity of AOP systems", International Journal of Innovations & Advancement in Computer Science (IJIACS), Volume 7, Issue 3, March 2018, ISSN : 2347-8616	Academic Science	2.65	Referred	Yes	**UGC Approved**
5	"Analysis of Reliability Model with the application of MCDM", International Journal of Emerging Technology and Advanced Engineering (IJETAE), Volume 8, Issue 3, March 2018, ISSN : 2250-2459	IJETAE Publishing House	6.351	Referred	Yes	**UGC Approved , ISO9001: 2008 Certified Journal**
6	"An Improved Model To Estimate Quality of The Software Product ", International Journal of Research, YMCAUST, Vol.1(II), July 2013, ISSN : 2319-9377	YMCAUST		Referred	No	

List of Published Papers : International Conferences

Sr. No	Title of the Paper along with Volume Issue No, Year of Publication	Publisher	Impact Factor	Whether Referred or Non Referred	Whether you paid any money for publication	Remarks
7	"Performance Efficiency Assessment for Software Systems", 50th Golden jubilee international annual convention of Computer Society of India (CSI-2015) theme Digital Life, organised by BVICAM New Delhi, 2nd to 5th December 2015 Print ISBN 978-981-10-8847-6 Online ISBN 978-981-10-8848-3	Springer			Yes	**CSI Sponsored** Proceedings in Software Engineering. Advances in Intelligent Systems and Computing, vol 731. **Springer**, Singapore **SCOPUS Indexed**
8	"Quantitative Evaluation of Proposed Maintainability Model using AHP Method", 9th International Conference on Computing for Sustainable Global Development (INDIACom), 2015, Organised by BVICAM New Delhi. Mar, 11-13 2015 ISSN 0973-7529; ISBN 978-93-80544-15-1	IEEE			Yes	**SCOPUS Indexed**
9	"Analysis of Quality Attributes and Metrics for Various Software Development Methodologies", International Conference on Advancements in Computer Applications and Software Engineering (CASE 2012), 21-22 December 2012, ISBN: 978-93-81583-77-7	Mewar University			Yes	

List of Communicated Paper : International Journal

S. No	Title of the paper	Name of the Journal	Present Status	Year
10	"Incorporating Supportability in Software Usability and its Assessment"	Indian Journal of Pure and Applied Mathematics (Springer) (SCI-Expanded, SCOPUS,UGC Approved)	Communicated (Under Review)	April, 2019