University of HUDDERSFIELD
Inspiring tomorrow's professionals

MEASUREMENT METRICS FOR HYBRID ASPECT ORIENTED/OBJECT-ORIENTED SOFTWARE SYSTEMS

By

Mazen Ismaeel Ghareb

Supervised by Dr. Gary Allen

School of Computing and Engineering

University of Huddersfield, England, UK

This dissertation is submitted for the degree of Doctor of Philosophy in Computer Science

December 2021

ABSTRACT

This dissertation proposes a new framework for the measurement of software product quality for hybrid systems. Hybrid system measurement is developed using a combination of Object Oriented Programming (OOP) and Aspect Oriented Programming (AOP) techniques. To make the required range of measurements for such hybrid systems, new metrics for AOP quality measurement will be proposed. These metrics will be collected from a range of open source projects using specific tools, and will be evaluated to prove their value. These metrics will then form part of a new software quality framework, used to evaluate the overall quality of the selected projects. This new framework is based upon the existing and widely adopted ISO 9126 quality model.

The outcomes of this thesis will contribute to hybrid systems' quality measurement. The theoretical contribution of existing software quality frameworks will be evaluated, and existing OO and AO metrics will be discussed. The new metrics will then be identified, and an expanded quality framework will be presented. I will then conduct intensive statistical analyses to evaluate the proposals.

The methodology of this thesis will be an experimental method to identify and evaluate AO metrics, combined with a statistical analysis to prove their efficacy.

The major contribution of the work will be to help developers and designers to improve their hybrid application systems in several ways, including quality, maintainability, efficiency, and complexity.

ACKNOWLEDGMENTS

I am too timid in all humility and gratitude to acknowledge my depth to all those who have helped me to put these ideas. Special thanks to my supervisor, Gary Allen, and his co-leader, David Wilson, and Diane Kitchen. They gave me an incredible opportunity to do this wonderful project called “MEASUREMENT METRICS FOR HYBRID ASPECT ORIENTED/OBJECT-ORIENTED SOFTWARE SYSTEMS”. Which also helped me to do a lot of research and I came to know so many new things. I appreciate that very much.

Any attempt at a certain level can only be made with the support and guidance of my parents (MOM and DAD) and my sisters (Dr. Hiba, Lana and Shahla), and my beloved brother Dyar for their continued support. I’d like to thank my wife (Rezhaw) and my two daughters (Mila, Zeen) who support me and encourage me to continue to finish this project. Despite the tough times of this global pandemic. Thank you all.

TABLE OF CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

AOP	Aspect Oriented Programming
OOP	Object Oriented Programming
AO	Aspect Oriented
OO	Object Oriented
SEM	Structure Equation Modelling
ISO	International Organization for Standardization
IEC	International Electro technical Commission
AOSD	Aspect-Oriented Software Development
ASOC	Advanced Separation of Concerns
SOC	Separation of Concerns
AP	Adaptive Programming
CF	Composition Filters
SOP	Subject-Oriented Programming
MDSoC	Multidimensional Separation of Concerns

LOC	Line Of Code
CMM	Capability Maturity Model for Software
SPICE	Software Process Improvement and Capability determination
CMMI	Capability Maturity Model Integration
SDLC	Software Development Life Cycle
CBO	Coupling Between Object
BAC	Base Aspect Coupling
C&K	Chidamber and Kemmerer
MSE	Modelling of the Structure Equation
AO/OO	Aspect Oriented – Object Oriented
SEM	Structural Equation Modelling
WOM	Weighted Module Operations
LOC	Lines of Code
DIT	Depth of Inheritance Tree
Code-MR	is an architectural software quality and static code analysis tool
LCO	Lack of Consistency in Operations
NOC	Number of Children
CIM	Coupling on Intercepted Modules
FEAT	Feature Exploration and Analysis Tool
CDA	Crosscutting Degree of an Aspect
WMF	Weighted of Method Factor

CDA	Coupling on Advice Execution
CDC	Concern Diffusion over Components
LCC	Lack of Concern based Cohesion
AC	Aspect Complexity Metrics

CHAPTER 1

1.1 Introduction

This chapter emphasises the goal and objectives of this study. It discusses the importance of the aspect-based approach to software development. This thesis will focus on finding new metrics for Object Oriented/Aspect-Oriented (OO/AO) hybrid application systems. The new measurements will provide a new software quality template for the OO/AO hybrid application system. The Quality Model is a new measuring model for hybrid OO/AO software systems. This thesis aims to offer a new quality framework for hybrid OO/AO systems. It also explains all stages of taking these additional measures into account in the proposed new quality model. This research will contribute to software quality and proposing a new product quality framework for hybrid AO/OO applications. Aspect Oriented Measurements of programming quality and product quality of the hybrid application system were reviewed. The framework will allow programmers and software developers to use this model to measure software quality for AOP hybrid systems (AOP, OOP). The primary aim of this research is to identify several quality metrics for AO/OO application product quality. There are many ways to improve the AOP quality parameters of the software. The ISO 9126 quality measuring framework has been adapted to extract potential parameters from hybrid application systems. The ISO 9126 standard of this research allows proposing a new framework for measuring the quality of AOP products for hybrid application systems. Static measurements shall be examined for the hybrid application system.

An AO product quality measurement framework will be proposed. The framework will present the effects of the new AO/OO measurements on hybrid software applications. It can also be applied to

any hybrid application project, whichever programming language or paradigm has used to develop it. This research has several aims, and these have been subdivided into objectives. One of the essential stages of new framework development is evaluating it. The same evaluation procedure adapted to the ISO 9126 quality model [1] can assess the proposed metrics. It has been proposed to add sub-characteristics to external quality measurements in ISO 9126. Several open source static measurement tools have been used to gather and evaluate existing metrics and it have been contributed to extracting new AO/OO metrics [2].

Crosscutting concerns are one of the main issues of Aspect Oriented Programming (AOP) which contribute to resolve it. It is essential to determine its impact on software quality. It employs new abstraction and compositional techniques for implementation methods. The fundamental problems identified the crosscutting concerns are the overall constraints of system properties, synchronisation, logging and error management. Crosscutting is divided into two types, static and dynamic. AOP improves software development in many fields, such as crosscutting implementation issues. There are many studies have conducted quantitative evaluations of AOP measures for software quality and software quality models, however these studies have not measured all AOP measurements because of several reasons, such as different in architecture, source code weight and development methodologies. .

1.2 Aims and Objectives

The aims and objective of the thesis is to propose several new Aspect Oriented (AO) quality metrics. To identify a new software product quality model for hybrid application systems using case studies and an extensive literature review method. Adding more it helps to identities and evaluate of AOP measurements in a hybrid system in many industrial applications, identify the research gaps related to aspect-based measures for hybrid application products. The objective is to add five major components to the proposed product quality model, such as extensibility, complexity, and service loading behaviours, code weaving, and reusability for ISO/IEC 9126 software quality model. Structure Equation Modelling has been used to evaluate the proposed metrics. Seven statistical measurements have been planned to verify the acceptance and reliability of quality model. Finally, the proposed framework will improve hybrid application systems regarding the reliability, functionality, efficiency, portability, and maintenance of the hybrid application system..

1.3 The Scope of the Study

Quality hybrid software applications have been adopted across a broad spectrum of development areas. Many hybrid application projects are necessary for evaluating and measuring aspect-oriented measurements. A number of measurement frameworks have been proposed to determine quality parameters for aspect-oriented applications in the hybrid application system or purely aspect-oriented applications. These AOP parameters were proposed in theory or logically, but neither the tool nor the framework collected them and proposed them as a unique quality framework. A standard version of the ISO/IEC 9126 software quality model has been selected to deliver AO measurements for hybrid application systems...

The objective of this study is to propose an evaluation framework for a hybrid application system involving the measurement of AO/OO metrics with a unique quality measurement framework. Therefore, it is critical to determine standard quality parameters between OOP and AOP through the development of taxonomy. This new framework will integrate a subset of each category into the baseline framework selection and then implementation. Finally, it must be evaluated, test the result, and make sure that can be generalised for development and expansion in software development industries.

1.4 Research Goals

The proposed framework needs to be evaluated and tested in several projects to show the hypothesis in the Goals and Objectives section. The major research goals are improving measurement of software quality and help researchers and practitioner measure different project and application developed by AO/OO technology. Employ ISO/IEC 9126 standard method and techniques in the software quality framework. It was necessary to propose new characteristics in the ISO/IEC 9126 measurements to identify the measurable elements. Improve the modularity of hybrid application systems by identifying the impact of AO code weaving applications. Enhance software product quality maintenance and software quality development by identifying AOP parameters in hybrid application software. Identifying the complexity of the system in the early stages of its development and controlled it by implementing the separation of concerns. Expand ISO/IEC 9126 framework for identification and evaluation of product quality metrics of hybrid application software. Decide how these techniques within this framework will be applied to identify aspect-based measures for software quality.

Propose a new classification for the proposed framework to determine AO metrics and explain the feasibility of the product quality hybrid application. Categorise a product quality measurement for the proposed quality framework for hybrid application systems. Determine the effect of common OO software quality metrics with the AO within the proposed framework...

1.5 Methodology

A detailed literature review and case study methods test the hypothesis derived from the design framework in parallel to a controlled experiment. This method is selected to protect the foundation of this research and to assess the framework of the proposed quality model, however, the study experiment could not prove all the claims made in this study, in particular all the AOP measures in the hybrid application system. Thus, empirical evidence cannot be achieved by itself without establishing it based on the theoretical method. Quality hybrid software applications many have been adopted in many fields of development. Many hybrid application projects are needed for the assessment and measurement of aspect-oriented measurements. A number of measuring frameworks have been proposed to determine quality parameters for aspect-oriented applications in the hybrid application system or purely aspect-oriented applications. These AOP parameters were proposed theoretically or logically, but neither the tool nor the framework collected them and proposed them as a unique quality framework. A standard version of the ISO/IEC 9126 software quality model has been chosen to provide AO measurements for hybrid application systems..

1.5.1 Requirements and analysis

This stage will focus on a qualitative analysis of the literature review and analysis of software quality modelling tools for the evaluation of the AO project. The research concentrates on the development of a new framework based on the AOP evolution of hybrid application systems. These steps will be evaluated, and new measurements will be proposed and applied in the context to be used to produce AO product quality measurements.

1.5.2 Design and Implementation stage

The design phase involves proposing new future and quality measures for developing the framework. By adding more, it will allow new techniques to implement a script or use the existing tool to evaluate and propose AOP product quality measurements in hybrid systems.

1.5.3 Evaluation

The evaluation stage involves the evaluation of the outcomes of the new metrics. Results from AO/OO measurements will be compared to traditional measurements for the hybrid application system. For example, an output generated by the new framework will be compared to a previous data analysis.

1.5.4 Experimental Study:

The experimental study used five free code projects developed using a hybrid AO/OO development approach. These projects were reviewed and assessed. The new results from the AO metrics will feed into the proposed framework for measuring hybrid application software. Finally, the results of these experiments will be compared with other literature review results if there is similarity of metrics to prove the proposed metrics.

1.6 Original Contribution:

There are many tools to evaluate AOP quality metrics with the various languages and programming frameworks available. Some of these frameworks overlap, but do not pull out all AO/OO quality measures. These methods have been implemented in many different ways, such as analyzing open-source code projects, highlighting code, or implementing scripting for metric extraction. The AOP quality parameters for software products will identify further analysis of hybrid source code projects. Several technologies were used to retrieve new AOP quality settings. The proposed metrics will not measure all performance metrics. There are many difficulties in calculating dynamic measures at execution, for example, linkage and cohesion measures; therefore, it concentrates on static measurement of the source code. Some of the dynamic measurement can be extracted because its characteristics will be affected by statistical measurements. A proposed framework is currently included in ISO/IEC 9126. The new framework has been extended by the addition of product quality sub-characteristics and a number of AO/OO measurements within the framework. In addition, the study analyses five hybrid case studies. Furthermore, the product quality hypothesis of the AO/OO hybrid application system, which will constitute a final contribution, was examined and demonstrated..

1.7 Published Works:

Journal Publication:

Ghareb, M. and Allen, G., 2015. Improving the Design and Implementation of Software Systems uses Aspect Oriented Programming.

Allen, G. and Ghareb, M., 2015. Identifying Similar Pattern of Potential Aspect Oriented Functionalities in Software Development Life Cycle. Journal of Theoretical and Applied Information Technology, 80(3), pp.491-499.

Ghareb, M. and Allen, G., 2019. An empirical evaluation of metrics on aspect-oriented programs. UHD Journal of Science and Technology, 3(2), pp.74-86.

Conference Publication:

Ghareb, M.I. and Allen, G., 2018, April. State of the art metrics for Aspect-oriented programming. In AIP Conference Proceedings (Vol. 1952, No. 1, p. 020107). AIP Publishing.

Ghareb, M.I. and Allen, G., 2021. Quality Metrics measurement for Hybrid Systems (Aspect Oriented Programming–Object Oriented Programming). Technique: Romanian Journal of Applied sciences and Technology, 3(3), pp.82-99.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This section summarises the software measurements. The chapter also deals with the parameters of dimension-based programming. It concentrates on the exploration of the documentary analysis of static and dynamic aspect-oriented measurements. The chapter focuses on the measurement settings of the software. It provides product measures, process measures, complexity measures and quality measures. The chapter also focuses on the history of object-oriented metrics in Champeaux [73], Common parameters such as complexity, modularity and size, depth of inheritance, coupling and cohesiveness. This can be used in Hybrid Application System measurements and this chapter explains the similarity and the difference between OOP and AOP metrics. There are similarities in programming techniques. Therefore, Aspect Oriented Programming metrics has been derived from several Object Oriented Programming metrics. There are two types of metrics in software measurement: Dynamic and Static metric. Static metric is measured software source code quality, such as cohesion measurement, which measures the relationships between classes and data in a module. Meanwhile, few studies have been conducted to identify dynamic metrics for AOP software [94] and Hybrid AO/OO systems, such as size, coupling, cohesion, separation of concerns, and depth of inheritance. AOP dynamic coupling metrics have been examined and evaluated at runtime execution. These studies have used Coupling between Object CBO metrics for evaluation [92] [93] [94] [95], these researches have not identified the AOP coupling effect on the software systems. According to [96], Coupling has measured by calculating different classes and identified the relation between objects at execution time. Object Oriented Programming metrics have increasingly recognised in measuring software quality, for instance, the error monitoring of the final system delivery [95]. Therefore, it is essential to identify AOP quality metrics hybrid AO/OO systems development.

2.2 BACKGROUND AND MOTIVATION

2.2.1 History of Aspect Oriented Programming

Aspect-oriented programming is a new paradigm that strongly influences software design, implementation, and quality improvement. AOP improves encapsulation by using Aspects, as each transversal problem is currently executed as a crosscutting module in a system. AOP extends another programming language, such as Java, or C++ for performing its basic techniques and mechanisms. For instance, Java developers use Java as the base language, which is important for the AOP system [4] [5]. In 1997, Gregor Kiczales formally revealed aspect-oriented programming [6].Several reviews, such as Walton, Allen, Ghareb, and others ([7] [8] [9] [10] [11]) have been conducted to find that it is possible to apply the metric measurements characterised for OOP on software developed with AOP. Not much work has been done to identify additional measures for aspect-based programming. According to Gulia and others [11], their research focuses on size measures, such as the number of classes in the project, their functions and the details of the source code. The Arisholm study [12] examined the costs of timeliness and the effectiveness framework. Ceccato and Tonella [13] used response time for their measurement, including estimation of cyclomatic complexity, coupling of elements, weight, and class relationships. Mitchell and Power [14] worked on collection measures in their evaluation of Aspects Oriented software applications. The focus has been on simplicity, viability, reuse and testability of quality components. The collection metrics measured the technical calls, the connection between the objects, the depth of the hereditary tree, the child numbers, the lack of cohesion of the methods and the reaction to a class. Kali and Kojarski [15] have utilised ten collection metrics to analyse Object-Oriented systems. Capsules' and Hilsdale [16] have reused view study that used collection metrics identify coupling in Aspect Oriented systems. Several other technologies and methods have been used to propose Aspect Oriented metrics. Research Li and Zhou [17] proposed software measures based on complex aspects through visualisation methods. The first stage defines the features and metrics of AOP programming to measure the general principle of quality. The research used visualisation methods to analyse the quality of a variety of size of software systems [17]. They have emphasised that qualitative analysis needs to be adapted in an AO format and apply in all related methodologies. Meanwhile, Rashid, Awis and Ana [18] came up with an AOP measurement tool to measure all the source code that is written in the AspectJ language. The tool is supported by a static text analyser developed in the TXL source translation tool. The tool comprises three modules;

- The first module was assigned all the source codes (Aspects, Interfaces and Classes)
- The second module developed into the reverse engineering code.

- The third module recognised the method calls among the source code objects.
- The last module is the most complex module. It gathered the weaving process of all Pointcuts and Join Points in the appearance code. The ultimate results showed aspects' effects on the system measurement of the quantity of aspect data transaction.

The AOP measurement tool offers a quantitative approach to improve the product development process and effectively. AOSD or ASOC is most focused on modularisation enhancements. The Separation of Concerns was brought forward by Dijkstra [23] and Parnas [24]. They have worked on how to separate crosscutting concerns from the traditional development process. Adding more there are many development techniques of modularisation developments such as Adaptive Programming [25], Composition Filters [26], Subject-Oriented Programming [27], Multidimensional Separation of Concerns [28] and Aspect Oriented Programming [29]. Khan and Nadeem [22] argue that as the number of modules increases, so does the hybrid application system. AOSD refers to the combination of separate ASOC methodologies. AOSD is genuinely new; therefore, it is necessary to propose a software quality framework for measuring AOSD and AOP developments methods. Various software development methodologies help developers organise, and control their development. AOP is one of those software methodologies introduced in 1996. Adding more, there are several approaches, identifying aspects of modelling improvement such as, viewpoints approach [19]; use case scenario-based approaches [20], theme approach [21], though these previous approaches still cannot identify all potential aspects. Crosscutting concerns techniques in AOP have been developed modularity for design patterns and programming techniques [24]. The design models of the development method are made method up of two categories. First part focuses on the search for aspects, classes, methods and other operations related to their components and the solution. Second part of the design concerns the structure and behaviour among the components. For example, the adapter pattern can implement both classes with the same interface to share the same operations or components; using AO techniques could be resolved using the same interface. The observed pattern works as defined by many dependent objects. One change happens to the observed variable; it will automatically notify another dependant variable, while when using AO techniques make the class familiar for the dependant and all notifications and make it notify for another part of the pattern [25]. The implementation phase of the design pattern involves many problems concerning inheritance, overlapping with the model and encapsulation. AO techniques have resolved all these drawbacks by using crosscutting techniques [26]. The addition of AO techniques increases the modularity of design model programming techniques, ensuring weaving between system components and making it more organised. Little researches have been done to illustrate the effects of AO metrics on software quality. According to

Jacobson and Ivar [18], an AOP development method that supports the software development to capture the domain related processes in the system, which claimed that can be better than OOP to fit real domain problems; the advantage is reducing debugging time. AOP also promotes a new behaviour of the code exchange of advice showed joints of the application while maintaining and support the independence of the framework from a major update in the design. There are accessible programming tools that support AOP, such as Aspect C++, JBoss AOP, Aspect#, Jasco, Spring AOP & AspectJ [29], [30], [31]. AOP has collaborated in three ways: code style, annotation, and XML [32] [33] [34] [35]. The main concern is that AOP techniques can improve object-oriented programming abilities by isolating concerns, adaptability, efficiency and maintenance [36] [37] [38] [39] [40]

2.2.1 Aspect-Oriented Programming principles

The primary aim of this study is to come up with a specific method or set of rules and regulations to discover all the quality metrics for AO/OO for hybrid application systems. Object-oriented programming techniques have issues in code tangling and code scattering. Implementing the AO focused on these issues and implemented cross-cutting concerns within a framework [3]. AspectJ supports all keywords and programming skills in Java. AOP components are offer greater modular capability [17]. AspectJ comprises key categories like JoinPoint, inter-type statement, Pointcut, Guidance and Aspect. There is also an issue with Aspect [16] [17], which is the way of modelling it in software projects. Advice is a piece of code invoked to fulfil a specific requirement. A Pointcut determines the invocation of the Advice. A Pointcut is a collection or group of Joins points. The JoinPoint are a special point in the programme's implementation [29]. These components may be described below:

JoinPoint: A JoinPoint is a specific, recognisable point in a program’s control flow, such as calling a method, executing a constructor. A method/constructor call is the most frequently used by JoinPoint. In addition, a field represents a specific property in a system and has two JoinPoint to call writing and playback. Pointcut: A Pointcut is a set of Joinpoint. A Pointcut may carry a name or an anonymous name. In almost all applications, a Pointcut is defined as shown in Figure 1 below:

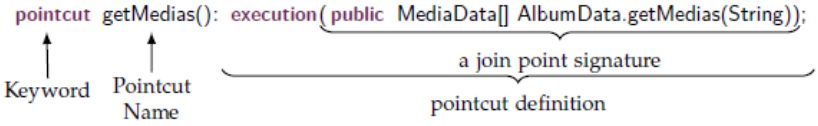


Figure 2.1 a named Pointcut

Figure 2.1 above showed Pointcut with the name NetMedia (), A Pointcut is used in many categories, like field get, field set, and method call and method execution [7].

Advice: It is action taken by aspect joint-point. The Advice also refers to a body of code. Advice like method has a declaration and body. Advice has to support the implementation of the crosscutting action. The exception techniques can be used in Advice and by using throws keyword, adding to this you can use this keyword, which refers to the appearance instance [7].

Intertype Declarations: The Intertype statement in AOP provides classes and interfaces with new functionality. The Intertype declaration is available in three formats: builder, method and field. Intertype members are only associated with their existing types and hierarchies and reported in aspect modules [7]. Figure 2.2 below shows that the colour is defined as a private member and that the colour type is included in the class of vessel of the appearance.

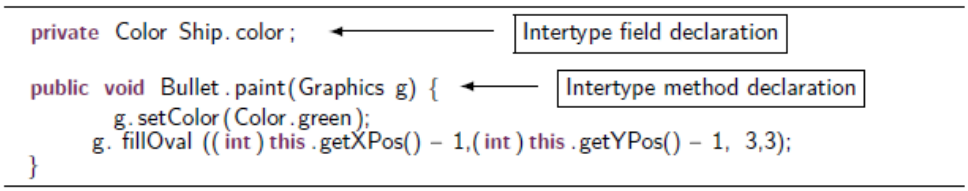


Figure 2.2 Example of intertype method and field declarations in an aspect module

According to Berkane [23], it is difficult to measure the quality of AO at the development stage. There is no quality measurement for all AOP metrics as a quality model framework. There are many types of research on test measurements in the design phase or in the development phase of a hybrid application system. Few standardised AO metrics were derived. The survey will continue to identify any quality measures. AOP has received the major control over modularity and source code organisation. AOP improves project quality through improved maintainability, portability and user-friendliness [27] [28]. For instance, in a software development model, the model-driven method improves the model structure, source code, and runtime [29]. The development of model-based software will improve the development structure, source code and runtime objects [29]; hence, AOP produces applications that have more reliability and flexibility in the development process.

2.3 Software Metrics Characteristics

Software measurement is essential for software engineering. Software measures play an important role in developing high-quality prediction systems for large software application projects

[52]. The measures are used to understand project maintenance and method for improvement of software development [53] [54] [55]. In addition, software metrics are key resources for measuring and institutionalising software process development in software-intensive settings, as shown in Figure 2.3. Software metrics are defined in several ways, as described below.

- Definition 1: Software measurements are used to measure the software production process. It provides the features involved in the product or quantitative values of the process. [58].
- Definition 2: Software measurements include specific quantitative explanations of attributes; these attributes are derived from the software product, the software creation process, and relevant resources [59].
- Definition 3: Software measurement offers a continuous measurement of the software production process and its associated products. It identifies, collects and analyses observable process data, thus facilitating the understanding, assessment, monitoring and improvement of the software product process [61].
- Definition 4: According to the "Quality Metrics method Software Standard" of IEEE, software metrics are a function of input as software, data, and output is a value that can assess how the attribute has affected the software [62].

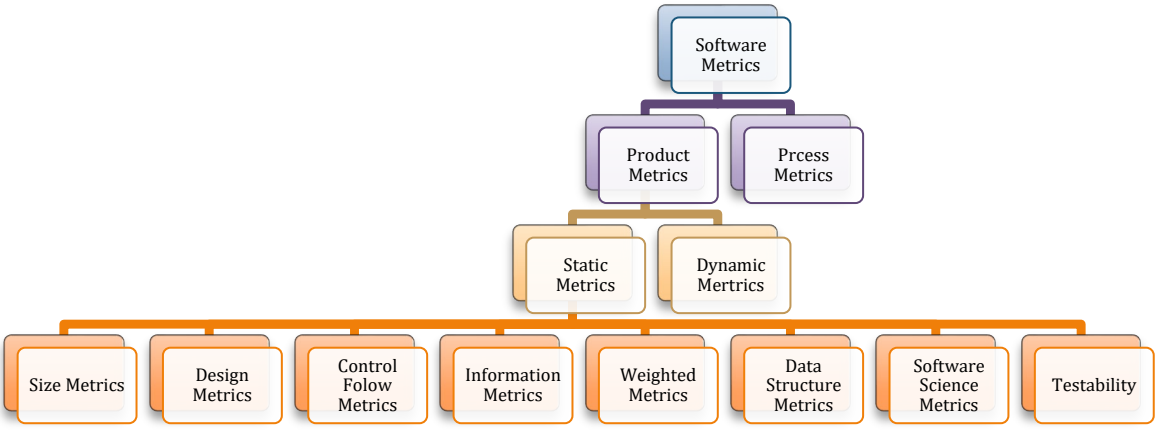


Figure 2.3 Classification of Software Metrics [62]

Programming metrics are important properties of the software systems that manage the developed product [10].

The fundamental characteristics of the metric are:

- Measurements need to be simple and definable.
- Easily accessible: Measures should be readily available.
- Valid: The measurement must be computed based on what is being measured.
- Robust: Measurement should not be performed for unrelated changes simultaneously on a program or part of the software.
- Success in the software development process requires assessment, estimation, and demonstration. Programming parameters provide a quantitative principle in the product

development process. The measures can improve the efficiency and quality of planning. Metric estimation models set out the most commonly used programming measures. It helps build templates with a high level of capacity in the execution time. Several products measures have been assessed and approved. They are in a state of constraint. The search for regularly agreed programming measures continues [6] [7]. Three categories of are used in the software development process.

- Product metrics: Documents and programs generated through software development. They measure the complexity of the source code, the size of the design and the number of pages of documents supplied.
- Process metrics: is a collection of software development processes, such as design, deployment, testing, maintenance,. An example of process metrics is an approach used for the average level or experience of programming personnel.
- Resources: support resources such as programmers, commodity and process expenditures [60] .Process parameters include the product development process.
- The Additional groupings of software measurements, aim measurements and subjective measurements are available. Objective measures shall constantly provide the same values for at least two qualified observers in a measure, while subjective measures are those which even qualified observers can quantify the changed qualities for a measure, since their subjective judgement is associated with the search for the measured value. For product measurements, the estimated Line Of Code article size is an objective measure. The case of subjective measures is the classification of programming as “individual” or “paired” as required in the cost estimate model [93]. The third approach to the organisation of programming measures is the primary measures and the calculated measures. The primary measures can be monitored legitimately, such as the LOC, the number of defects in unit tests, or the aggregate development time for the task. Not calculated measurements cannot be observed directly, but are calculated from other measurements [8]. Programmatic parameters are used in developmental engineering approaches. It is critical to take further action if some elements of software are incorrect in the specifications given before or during maintenance of the product support. Some of the standard parameters are discussed and considered for a relatively long time; however, few are taken into consideration for experimental work. Software engineers should be able to rely on tools that update these measures to assist them in quality assessment and assertion missions in order to measure the quality of programming. Today, many metric programming instruments are accessible; however, it would be critical for each measurement mechanism to perform the recommended set of measures and how they were approved. Adding more, dynamic and

static metrics are two types of software assessment. Static metric is measure the quality of the software source code, such as cohesiveness, which measures the strength of the relationships between the data and the module classes. While dynamic metrics measure the relationships between different software components at execution time.

2.4 Software Measurement Models

Software measurements are a cornerstone of initiatives, including CMM, ISO/IEC 15504 SPICE and CMMI. The ISO/IEC 90003:2004 standards [56] focused on the importance of measurement in quality assurance and management. Given all the efforts and improvement over the last decade in research and international standardisation, software measurement is still defining, standardising, and determining definitions, principles, and methods. As mentioned above, there were many ISO models for software measurement, but there is no clear framework for measuring AO/OO hybrid applications.

2.4.1 Product Metrics

Software product measures are measures of software products like source code and document design. Software design influences and affects the process parameters. Product metrics are part of software measurements which measure software products within different paradigms. It has been measured throughout all stages of software development, from requirements to software application [57]. The product metrics are divided into two categories: static measures and dynamic measures. Static measurements are taken from system representation, like design, documentation, and program code. Dynamic measurements are gathered from the execution of the program. There are various measures of software products, such as:

- Fan In/Out: Fan I/O denotes the number of functions that call for other functions. Fan-out means the number of functions called by the fan in. A high fan-in value means that it is tightly coupled with another part of the design and has a significant effect on the related components. A high Fan-out value shows that the components need more coordination within the calling function to reduce complexity [57].

- Length of code: measures the size of the program. The enormous size of the code results in more complex software.

- Variable length: Measurement of variable length. The longer the variables, the better the program can be understood.
- Cyclomatic Complexity: A measure of program complexity control. Monitor complexity related to program understanding.
- Fog Indicator: It's a measurement of the length of words and sentences in software documents. The higher value of the fog index shows difficulties with document comprehension.
- Conditional nesting of control statement: This is a measure of the breadth of the conditional control statement. High values of nesting control statement it leads to difficulty understanding of the code and increased complexity.

2.4.1.1 Static vs. Dynamic Metrics

Static measurements can be performed during the first stages of the SDLC. These parameters consider the structural characteristics of the software system and help to understand complexity of the system. However, dynamic measurements help assess the reliability and efficiency of the software. Static measurements of complexity may show how much effort is needed to develop and maintain the source code. The first static measurement is the code line [48] (LOC/KLOC) which is used to measure a programmer's productivity. The most common measure of complexity employed since 1990 is the cyclomatic complexity measured by McCabe [49]. McCabe used flowcharts and several mathematical formulae to calculate software complexity. This technique was used for risk analysis of code development, maintenance, and test planning. Static measurements are derived from a non executable source code analysis. Dynamic measurements are associated with the quality attributes of the software. It is possible to dynamic metrics at run time. They tell you which calls occur, how many instructions are executed, and which parts are executed. Dynamic metrics include complexity metrics and metrics useful for reliability modelling and directly related to software reliability. Table 2.1 provides some examples of software static measurements for object-oriented programming and some common static measurements for AOP and OOP.

.

Static Software Metric	Description
Attribute Hidden Factor (AHF)	This measure is used to measure invisible attributes within classes. It measure total classes whose quality is invisible.
Attribute inheritance factor (AIF)	The attribute inheritance factor is calculating the total number of inheritance attributes in all classes of the system.
Average Depth of Paths (AVPATHS)	Average depth of paths is calculated by counting total numbers of paths and size for all methods, dividing by the number of calling methods in the system. .
Average lines per method (AMLOC)	Average lines per method: it calculates by measuring the average size of the methods.
Depth of Inheritance tree (PDIT)	Depth of Inheritance tree: is calculating the deepest or maximum of all inheritance trees within the project
Lines of code (LOC)	Lines of Code: is calculating number of lines, white space and comments.
Method Hiding Factor (MHF)	Method Hiding Factor is calculated by summing the visibility of each method the project. Invisibility of a method means the proportion of total classes whose method is invisible.
Method inheritance factor (MIF)	The method inheritance factor provides information about the impact of inheritance within your file or program. It is calculating by finding total number of relationship between inherited methods and the total number of methods.
(Number of Classes in Program) NCLASS	Counts the number of classes in a program.
SEI Maintainability Index (SEIMI)	These measures make it possible to determine the level of maintainability of the software project.
Source lines of Code (SLOC)	Counting lines are used to estimate the amount of maintenance required and can be used to normalize software source code.

Table 2.1 Software Static Metrics

2.4.1.2 Complexity Metrics

Software complexity measures are a calculation of the costs associated with developing, maintaining, and using software. Complexity is the most important factor in the origin of defects. The reliability problem is mainly related to the complexity of the software. When complexity exceeds some levels, software gaps or vulnerabilities are increasing. The maintainability of the software is also inextricably linked to complexity. Software complexity analysis is the fundamental aspect of software assessment, which is the major method of software quality assurance.

Ghareb and Allen [68] summarise the six reasons behind the software.

1) System interpretation difficulty.

2) Difficult to repair bugs and maintain software.

3) Difficulty defining software for other users.

4) Difficulty in updating the software according to the guidelines.

5) Writing program workload based on specification.

6) Availability of tools required for program delivery.

Complexity measures will assist people in expecting and manage projects. The more complex the modules, the more effort they would have to put into testing and maintenance [68]. For complicated modules, we will see how to break them down and therefore their level of complexity, making the module easier to test and maintain. Complexity measures can aid in determining programming workload and development costs. It may support the assessment of the maintenance effort required. In addition, Complexity Measures can help with the same functions to select the most appropriate program. Complexity measures may also identify defects or mistakes.

2.4.3 Process Metrics

Computer process indicators are indicators of the process by which code is created. Steps in the software development process are steps. Examples of process measures include the average level or experience of programming staff used as a specific development method. There are many sorts of process measures. For example, COMM measures the number of engagements undertaken on a file. ADEV corresponds to the number of developers who changed the script. DDEV is the accumulated number of separate developers who contributed to the file until this update. ADD and DEL are the rows added and deleted from the file that are normalised (by the total number of rows added and deleted). OWN calculates the number of lines that the most significant writer of a file has written. Minor tests are the number of authors in the files who wrote less than 5% of the code [4]. EXP tests the file developer's highest experience using the percentage of lines he wrote in the project. EXP checks the geometric average of all developer interactions. All these measures are based on previous investigations [63] [64] [66] [67]. The SCTR measures the dispersion of changes in a file; scattered changes can be more difficult to monitor and therefore more likely to cause defects. SCTR is the standard deviation of geographic centre shift location. This metrics are used to assess the efficiency and performance of different processes.

2.5 Object Oriented vs. Aspect Oriented Metrics

There are various models for assessing the quality of object-oriented programming that are explained by [33] [34] [35] [36] [37]. Few projects measuring the quality attributes of the AO system have been carried out. The above studies have been proposed and assessed for many quality characteristics, such as complexity, coupling, reusability, modifiability, maintainability, cohesion. Burrow [44] provides an accepted and referenced method for object-oriented metrics. Some of the AO measures employed the same measures as the OOP. The Study has showed of relationships and

weaving process between classes, Aspects and methods, advice which can help determine other metrics of AO [45].

AOP separation of concern requirement is used to address scope requirements and constraints in software developments and it consider as an effective way to identify system requirements. Many approaches are used to design and implement the separation of concerns in AOPs, such as viewpoint. Tsang [42] proposed separation of the specification in AOP measures, such as compositional rule, aspectual requirements and non-special requirements in the design measures of the module. The compositional rule is used as informal procedures to identify how an individual requirement control or limits the performance of non-disciplinary requirements. The separation of the specification of aspectual requirements makes it possible to set up the separation of concerns in the first stages and may affect components in process later. Tsang used the Viewpoint approach and the extensible XML mark-up language to develop a tool to uncover the engineering aspectual requirements [42]. There are many similarities between the object orientation and the aspect orientation, in terms of class, methods, aspects, and advice. It must develop and propose a new AO measure based on OOP measures [46]. Quantitative evaluation was completed for quality, modularity attributes for AOP and OOP. The characteristics can be listed below:

Line of code and scattering. The evaluation was completed for each module and concern. Quantitative parameters such as size, dimension and crosscutting concerns. OOP metrics measure is used to evaluate the OOP source code development techniques in the source code. AOP measurements would have a negative effect on performing all systems [43]. For example, the depth of the inheritance will measure the longest path from the root of the Aspect hierarchy to a particular module in the system. The increase of identified the Aspects the more the transaction could be inherited in the system. As a result, it is more complex to understand and analyse the software [47].
Complexity and size: is measured by calculating total complexity of all functions and methods in the module. Classes are assumed to be created as fewer codes to enhance their complexity measures. Size metrics can be measured by size of methods and number of attributes for each class [48].
Cohesion: It has measured through class-level measures and calculated according to a different approach, it can be measured by finding relationships between functions and methods. There are three class-level measures, such as loss of class cohesion, lack of method cohesion, information-based cohesion and tight class cohesion [49].

- Re-usability: Re-usability measures are determined by the reuse ratio and the specified ratio. They are calculated by measuring the relationship between sub-classes and all other classes

and by measuring the super classes separately. The classes have therefore been re-used several times; general reuse measures are favoured [50].

- Polymorphism is measured by polymorphism factor and the number of methods exceeded by the class. This measures at different levels. For instance, if you need to be calculate polymorphism in a single class polymorphisms, it measures the methods calls in the class and the polymorphic factor measures excellence across the system [51].
- Encapsulation is used to attributes masking factors and method masking factors and shows how the attribute and method are well hidden inside the package. This measurement consider as system level measurements.
- Coupling: it measure by a calculating number of the coupling factor which measured at the frame level. The other factors are on the class level. The response of message transmission to the class. Coupling between objects means the coupling between class instances and data summary [41].

Several papers examine dynamic performance coupling metrics; they depend on CBO metrics [1] [2] [3] [4]. According to [5], who measured the coupling between different classes at execution dynamically, the metrics of the object coupling have been found only at execution time. The measurements proved to be indicators of vital quality properties of the final product of the system [4]. For CBO measurements, observational investigations were established for linkage measurements [4]; the results showed they were similar in several features. There are many examples of research projects on AOP metrics that have been conducted ([38] [39] [40] [41] [42]). This research concentrated on the application of characteristic measures for OOP. There is no framework for determining all (AOP) actions. Mickelsson [40] worked on size measures, counting the number of classes, functions. Coady and G. Kiczales [41] proposed a framework for identifying execution values and the position of hidden concerns. Zhang and Jacobsen [39] used Cyclomatic Complexity, Class Weight, Coupling Amongst Aspects, and measured the time for their assessment. Tsang [42] used Chidamber and Kemerer C&K's suite of measures to evaluate AO systems. They measured the components of quality, such as comprehensibility, Feasibility, Re-usability and testability. Zakaria and Hosny [43] explained the impact of the C&K sequence on aspect-oriented programming, and found many common indicators between OO and AO measures. Burrow [44] examined ten common metrics parameters between AOP and OOP. These measures were characterised in the Burrow and Garcia [45] series of measures and established object-oriented measures proposed by C&K's measures. Burrows and Garcia [45] proposed a new metric which assesses the coupling between the base source code and the Aspect code, called BAC. Burrows and Garcia [45] used C&K parameters for their experience and concentrated on identifying coupling in

systems developed by AOP. Dhambri [46] suggested a graphic visualisation technique to show the Aspect-Oriented metric of the software. The first stage defines the features and measurement of AOP programming. They used third-party applications to retrieve AO measurements based on OOP measurements [46]. The study identified metric definitions and suggested methodologies of value for aspect-oriented quality measures. Ceccato and Tonella [47] have developed an AOP measurement tool that measures software applications written in the AspectJ language. The final stage of the analysis focuses on creating a report showing the system's proportion affected by aspects and the amount of the modules they crosscut.

C&K [76] defined some metrics as listed below:

1. WMC: Weighted methods per class
2. DIT: Depth of Inheritance Tree
3. NOC: Number of Children
4. CBO: Coupling between object classes
5. RFC: Response for a Class
6. LCOM: Lack of cohesion in methods

These metrics can be used as common measurements for hybrid AO/OO applications. It has been evaluated and tested in chapter 5.

2.6 Software Quality Measurements

The Software development becomes more efficient, and the quality of software delivery becomes elevated to meet customer requirements after the year 2000 [9]. The measurement of software plays an important part in two drivers of software engineering efficiency and effectiveness. Software Quality Metrics deals with measurement attributes related to the process of development and the quality of software. Software quality measurements comprise these parameters: reliability, integrity, usability, maintainability, customer satisfaction, etc. [9], as it appears in Figure 2.4 below.

.

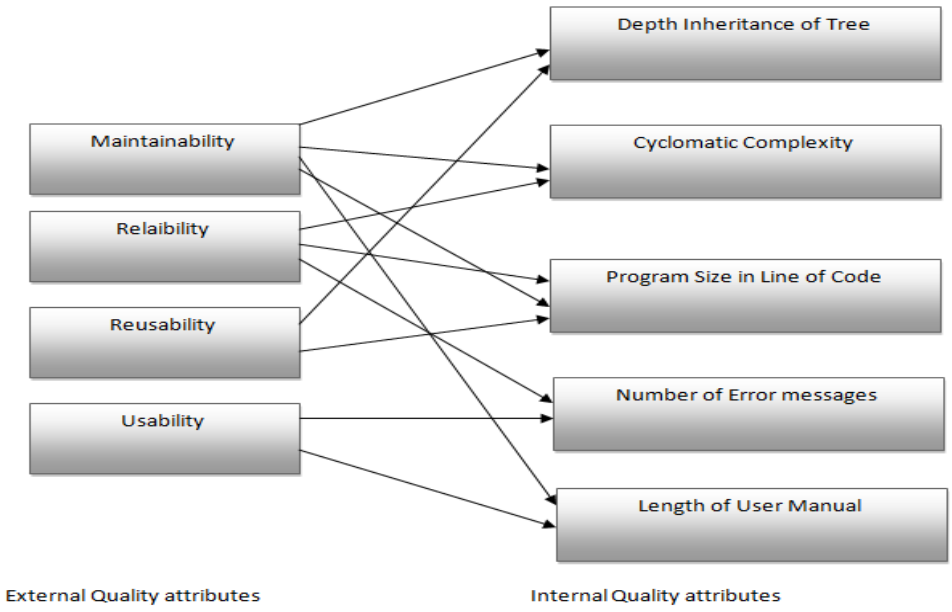


Figure 2.4 Software Quality Metrics

Quality metrics are a subset of software metrics that focus on product, method, and project quality aspects. They are closer to the method and measurement of products than to project measures. The efficiency of the software tests the way the software meets its needs. The specifications for applications are functional or non-functional [69] [70] [71]. Software quality measures comprise three major groups of measures: software product quality, process quality, and project quality.

Figure 2.5 shows some features of Product Metrics, Process Metrics, and Project Metrics.

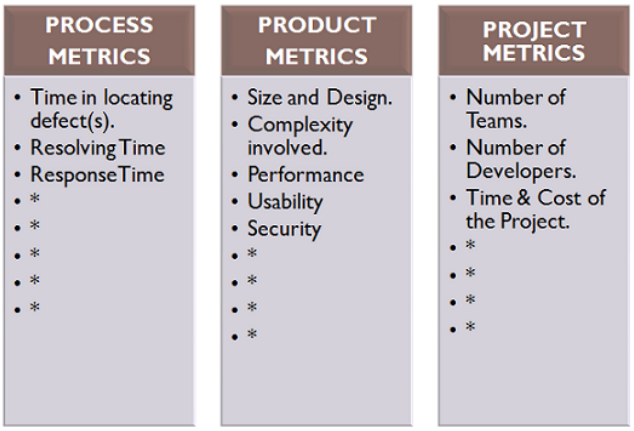


Figure 2.5 Features of Software Quality Metrics

These measurements are gathered across the project. It measures the process according to the specific attribute to develop metrics for those processes. Process outputs affect metrics based on this process, for example, completed on schedule, follow through with each activity, apply error management prior to product release and poor communication with the final user. Project metrics: It

covers the cost, the duration of the project, the number of developers, the number of teams, and so on.

2.7 Software Quality models

2.7.1 McCall Quality models

The McCall [130] model was incorporated with varying factors. Software factors affect the software. The model was classified under two categories:

- Quality factors: This quality factor is directly accessible and is regarded as a high-level quality factor. These are external attributes that both manager and users are concerned about.

  Quality Criteria: Assigned quality that is subjectively and objectively accessible. These attributes are internal attributes considered as a second level of quality attributes. The software requirements are classified into 11 factors, as shown in figure 2.6. These 11 factors are divided into three major commodity factors: Product Operating Factors: it concerns to the functioning of the product, such as accuracy, reliability, effectiveness, integrity and ease of use. Product Review Factors: it concerns with product review, such as testability, maintainability and flexibility. Product transition factors: it concerns with the transition of products such as reusability, transferability, and interoperability.

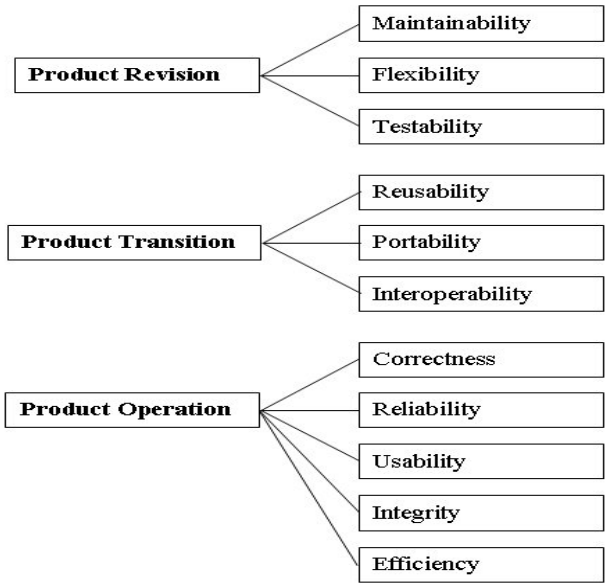


Figure 2.6 McCall software quality model

2.7.2 Boehm Model

Boehm [131] has a similar model to McCall. It represented its quality model as a hierarchical tree and decomposing quality features into several components, as it shows in Figure 2.7. The Boehm model concerns the broader attributes of the features that affect the quality of the whole software. The difference from McCall is evaluated the performance of the hardware features [132]. The Boehm model comprises three levels of quality attributes at primary, medium, and higher levels.

The Higher level comprises:

- As Is Utility: This is a software extender.
- Portability: Software upgrades based on new environments.
- Maintainability: Error detection and correction when performing maintenance.
- The following tier contains seven quality attributes:
- Modifiability: Software modification issue during maintenance phase.
- Intelligibility: User requirement to recognize logical concept and applicability.
- Testability: Requires verification of software functionality.
- Usability: It's about using, learning and understanding the software function.
- Effectiveness: understand the amount of code and hardware resources necessary to perform software functions.
- Reliability: it works on software according to the requirements.
- Portability: obligation to change the software according to new environments.

The last level in the Bohem model is to classify attributes into other primitive attributes such as "independence, accuracy, completeness, consistency, efficiency, accessibility, communication, self-reporting, legibility, structuring, brevity, ability to increase". The benefit of this model is a low maintenance cost and satisfying the needs of users.

Figure 2.7 Boehm quality model

At the end of the 1970s, McCall et al. [130] and at the end of 1980s, Boehm [131] proposed a framework for quality, which can help project managers to monitor risk and assess quality of the software. At that period, OOP technology was sufficiently developed. ISO/IEC 9126 [142] quality standards model has been proposed in the early 1990s, with a focus on maintainability. The heritage property of object-oriented technologies has emphasised reuse [143] has added reusability to the six features of ISO/IEC 9126. IEEE has released the Software Engineering Body of Knowledge [133], which was the first attempt for collecting, classifying and standardising the terms models, methods and techniques.

2.7.3 (9126) Software Quality Model

This model us deals with three principal components of software quality [1], such as,

(i) Process quality

(ii) Product quality

(iii) Product quality

It comprises six major quality features with a hierarchical model. These quality characteristics are broken down into 21 secondary characteristics which support to the internal quality. ISO/IEC 9126 1 focuses on defining quality characteristics and sub-characteristics of the final products. ISO/IEC 9126 2 have been worked on six external quality features to extract these external metrics. These features were defined by the ISO/IEC 9126 1 standard. ISO/IEC 9126 2 is set out for external measures and ISO/IEC 9126 3 sets out internal measures. ISO/IEC 9126 4 define quality through measurements for measuring quality features or sub-features. Internal metrics measure the software itself, and external metrics measure the computer-based system behaviour that informs the software for attribute updates. The quality metric measures the effects of using the software within a particular usage context. These are set out in Figure 2.8.The ISO/IEC 9126 2:2003 standard used with the ISO/IEC 9126 1 standard. ISO/IEC 9126 4 provide: An explanation of the application of software quality measures. A core set of measures for each feature. Example of measurement application throughout the software life cycle, however, the ISO/IEC 9126 quality model is fairly complicated but does not cover some important quality features, contributing to the quality of the AOP.

Quality Type	Characteristics	Sub-characteristics
Software Product Quality	Functionality	Suitability
		Accuracy
		Interoperability
		Compliance
		Security
	Reliability	Maturity
		Fault tolerance
		Recoverability
	Usability	Understandability
		Learn-ability
		Operability
	Efficiency	Time behavior
		Resource behavior
	Maintainability	Analyzability
		Changeability
		Stability
		Testability
	Portability	Adaptability
		Install-ability
		Replace-ability
		Conformance

Figure 2.8 Software Quality Model Properties for 9126

2.7.4 Dormer's Model

Filman, Elrad and Clarke [7] have defined a quality model which presents a simple process to build quality properties into the software. The model links the characteristics of actual products with those that are less concrete. The template has supported finding shortcoming and specifies properties of the software quality. This model addresses product quality by defining all related sub-features merged into higher-level features. The model supports integration of quality, development of automated codes for quality flaw detection in software definition of language specific coding standards, Reusable is a quality characteristic of its model, and the systematic classification of quality limitation.

2.7.5 Other Software Quality Models

Ghareb and Allen [10] have categorised software quality as the quality of software processes and products. Several characteristics of the software product development process are technology, personnel, organisation, tools, equipment, document clarity, integrity, design traceability, program responsibility, test integrity and organisation. In this model, guidelines were provided on the procedure for obtaining the desired product Gulia, Khari, and Patel [11] suggested that software quality measures may include:

- User based: The quality of the software evaluated by the uses and refers to satisfaction of the users' expectations.
- Product delivery: is referred to evaluation of the product delivery, system efficiency and program maintainability.
- Manufacturing based: the development process, insisting on quality control and management.
- Organisational control: it evaluated by project costs, risk management, production time and control of resources.
- Arisholm, Briand, and A. Foyen [12] defined a quality model in component based software development (CBSD) and derived their model from ISO/IEC 9126. CBSD model has been used for tracking and flexibility, reuse and complexity. It comprises six characteristics to assess the total quality of a component. Analytical Hierarchy Process (AHP) approaches have been used for measurements, and for the weight values of the quality characteristics and sub-characteristics. Chang et al. [13] provided guidelines for assessing software quality by integrating fuzzy theory and AHP. Their implementation is bringing new concept in the ISO/IEC 9126 quality model. Rather than gathering sample data, they applied a fuzzy theory to get relative weights of metrics values for all components.

2.7.6 Aspect-Oriented Software Quality Model

Allen and Ghareb [126] analysed an extension of ISO/IEC 9126 quality model. In this quality model, five sub components added to the framework, for instance, modularity has been added as sub components under maintainability.

- Code reducibility has been added as sub components under efficiency.
- Complexity has been added as sub components under usability.
- Reusability has been added as sub components under functionality.

- The new model is given in Figure 2.9. In this table, the components and sub component were also labelled as C2 in the following sections. The proposed model shows additional sub-features in the new framework of the existing ISO/IEC 9126 software quality model [151].

Quality Type	Characteristics	Sub-characteristics
Software Product Quality	Functionality:C1	Suitability:SC11
		Accuracy:SC12
		Interoperability:SC13
		Compliance:SC14
		Security:SC15
		Reusability:SC16
	Reliability:C2	Maturity:SC21
		Fault tolerance:SC22
		Recoverability:SC23
	Usability:C3	Understandability:SC31
		Learn-ability:SC32
		Operability:SC33
		Complexity:SC34
	Efficiency:C4	Time behavior:SC41
		Resource behavior:SC42
		Code-reducibility:SC43
	Maintainability:C5	Analyzability:SC51
		Changeability:SC52
		Stability:SC53
		Testability:SC54
		Modularity:SC55
	Portability:C6	Adaptability:SC61
		Install-ability:SC62
		Replace-ability:SC63
		Conformance:SC64

Figure 2.9 Aspect-Oriented Software Quality Model

2.7 Aspect Oriented Programming Quality Metrics:

Aspect-oriented programming quality metrics are the properties of the software that affect the primary functionality of the software. The major objectives are to improve the reuse of the module and the design of the software. Several studies have focused on software quality, measurements, and measuring AO [10]. Gulia and Khari [11] proposed five suites of measures based on a software model based on the aspects. The model comprises dependent charts representing various dependent relationships at a different level. Coupling parameter represents dependence among classes and aspects. The aspect cohesion measure tells you how closely the character modules (method and guidance) are logically coherent [11].Rashid and Awais [18] suggested extending the ISO/IEC 9126 software quality model. In the opinion of the researchers, the quality model would improve modularity. The improvement includes adding maintainability, code reduction, user complexity, and reuse sub-functions. The suggested model is presented in Table 2.1. In this table, all details of the highlighted properties are the new attributes offered. Definition and assessment are essential for initial and primary characteristics and sub-characteristics (ISO/IEC 9126) in AO technology. The model is referred to as the Aspect Oriented Software (AOSQUAMO) Quality Model;, however, this model does not include all AOP measurements related to AOP software quality and the evaluation method and case studies were not enough to prove it.

Software Quality Model ISO 9126		**Software Quality Model (AOSQUAMO)**	
Main Characteristics	Sub-Characteristics	Main Characteristics	Sub-Characteristics to be added
Functionality	Suitability	Functionality	Re-usability
	Accuracy		
	Interoperability		
	Compliance		
	Security		
Reliability	Maturity	Reliability	
	Fault tolerance		
	Recoverability		
Usability	Understandability	Usability	Complexity
	Learnability		
	Operability		
Efficiency	Time Behaviour	Efficiency	Code reusability
	Resource Behaviour		
Maintainability	Analyzability	Maintainability	Modularity
	Changeability		
	Stability		
	Testability		

Portability	Adaptability	Portability	
	Intsallability		
	Replaceability		
	Conformance		

Table 2.1 Quality Properties for 9126 ISO Models, 2004. Vs. New Aspect-Oriented Software Quality Model

Based on [14] [15] [16] [17] stated that, coupling is a dynamic metric for AOP, and this depends on other static metrics. Low couplings and High cohesive are a good standard software design, according to Kiczales [16]. Rashid [18] supports the idea of coupling metrics, which used for measuring AOP and OOP. The above study shows that AOP metrics can use same as OOP metrics because there are many similarities between the two technologies. They have suggested new measurement techniques for AOP; however, there is no single AOP measurement to measure AOP and OOP in hybrid application software. An ISO/9126 quality model as standard quality framework has been used [50]. ISO/IEC 9126 includes many categories; each category has several features and functionalities. As shown in Figure 2.10 below:

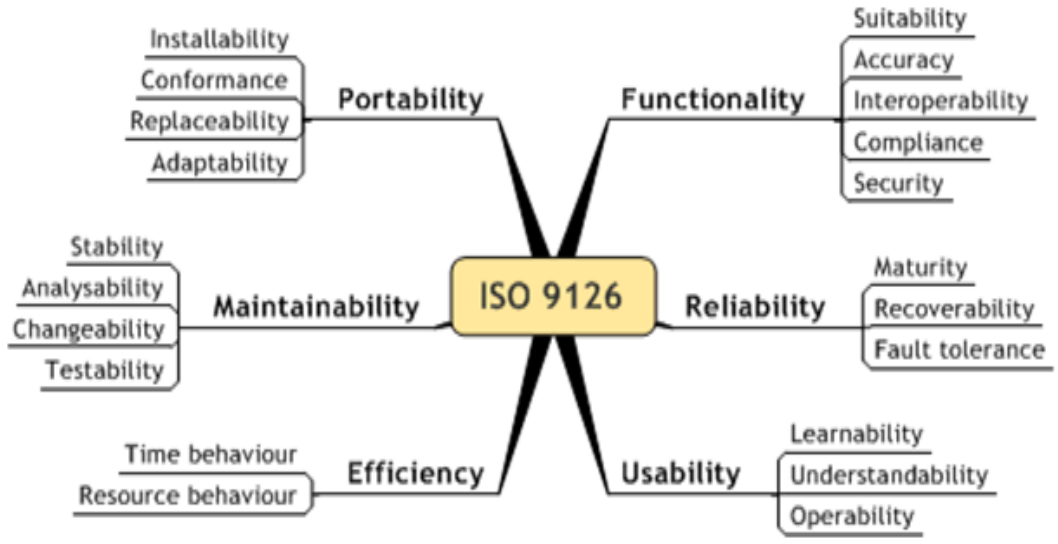


Figure 2.10 Product quality 9126 Model

Aspect Oriented quality metrics can be used along the same lines as software quality metrics. These categories have been implemented in ISO 9126. For instance, for functional features adding

scalability, in reliability features adding complexity, the service load behaviour was added for efficiency features. In addition, maintainability, code weaving, portability and re-users have been added. A quality measurement framework is not available for AO/OO systems. Software complexity encompasses multiple software properties that affect all internal interactions. The measures of those characteristics determine the complexity of the code. Software coupling corresponds to the level of interdependency between software modules. Reusing serves is to reduce the size of the software project code. Maintainability is the modification of the programming element after delivery. Cohesion is the level of relations between the components of the modules. There are various programming measures for these attributes. The programming measures were calculated using quantitative methods. Some measures for evaluating the quality of hybrid AO/OO system development have been proposed. This will help build a new software quality framework. Table 2.2 has been explain all the technologies and techniques that have been proposed by different authors. It has shown there are relationships between AOP and OOP metrics and they affect each other sometimes have negative effects and in other have improvements in the software metrics. The proposed hybrid quality metrics model can measure both of AOP and OOP metrics and find the relationship between them. This finding is to help developers, designers and project managers to choose appropriate techniques during development process.

Metrics Measurements	**Object Oriented**	**Aspect Oriented**	**Common Metrics**	**Remarks**
Line of code [6][7][43][47]	+	-	Yes	Line of Code has positive effect on OOP source code to evaluate it. AOP has negative effect on the performance of the all system.
Complexity and size [48][49][67][68]	++	-	Yes	The Complexity and size it depends on size of the project, generally this AOP it more affect if the complexity is high they effect negatively on software performance, however for small project it not effect on the size and code complexity.
Process Metrics[63][64][65][66][67]	-	+	Yes	This metrics it affect performance and efficiency the AOP metrics is more efficient than OOP.
Cohesion [49]	-	+	Yes	AOP metric is higher than OOP metrics here because OOP have different levels class and method cohesion.
Coupling [41]	+	+	Yes	In both technologies AOP and OOP coupling metrics values is vary. In both have disadvantages and advantages so it depends on coupling between class and object and data movements.
Re-usability [50]	+	-	Yes	AOP metrics value is higher than OOP metrics in majority of application AOP metrics improve the reusability.

Polymorphism [51]	+	-	Yes	In OOP it measure across classes while in AOP it measure across classes the execution of the methods or joint point. The metrics values of AOP are much higher than OOP.
Encapsulation [51]	-	+	Yes	OOP metrics is much higher value than AOP metrics on the system level.
Weighted methods per class / aspect [1][2][3][4][5][38][39][40]	+	+	Yes	AOP metrics is depending on OOP metrics.
Depth of Inheritance Tree [1][2][3][4][5][38][39][40]	+	-	Yes	OOP metrics values are higher than AOP metrics values.
Number of Children [1][2][3][4][5][38][39][40]	+	-	Yes	OOP metrics values are higher than AOP metrics values.
Coupling between object classes[1][2][3][4][5][38][39][40]	-	+	Yes	AOP metrics is depending on OOP metrics.
Response for a Class[1][2][3][4][5][38][39][40]	+	-	Yes	OOP metrics values are higher than AOP metrics values.
Lack of Cohesion in methods[1][2][3][4][5][38][39][40]	-	+	Yes	AOP metrics is depending on OOP metrics.
ISO 9126 Quality Metrics-Functionality [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
ISIO9126 Quality Metrics-Reliability [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
ISO 9126 Quality Metrics-Usability [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
ISO 9126 Quality Metrics-Efficiency [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
ISO 9126 Quality Metrics-Maintainability [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
ISO 9126 Quality Metrics-Portability [14][15][16][17][18]	-	+-	Yes	Quality metrics have been measured both AOP and OOP metrics but it not support all together as one values this is the disadvantages
Proposed Hybrid Software Quality model	++	++	Yes	The proposed Hybrid Software quality metrics models have advantages over all other model it measure all metrics of AOP and OOP and have been shown the relationships effected over the metrics.

Table 2.2 AOP and OOP metrics relationships in different model and framework

2.8 Summary

This chapter clarifies the software metric properties. It also provided a systematic literature review analysis of the most recent contribution to the measurement method. The AO and OO metrics discussed and defined the similarities and differences between them. Several methods for the determination of AOP measurement methods have clarified the benefits of collecting AOP measurements. The most important element in software engineering and software development is software measurement. The software metrics were addressed in all major categories, such as products, processes, complexity and quality metrics. In software measurements and software development, the importance of AOP measurements has been outlined above. It is essential that a single framework is provided to measure AOP and OOP quality metrics in hybrid application software.

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Introduction

This chapter will cover the research methodological techniques used in this research. Two research methods were used to conduct the research. A detailed review of the documentation to identify gaps in the existing approach to identifying AOP actions. A second method comprises a case study to propose a quality framework for hybrid AO/OO software. The primary aim was to establish static measures for the proposed quality framework in order to measure the development of hybrid AO/OO systems. Five case studies were then analysed and evaluated and compared to the proposed metrics for the AO hybrid application system's quality framework. In the research method chapter, it has been explained many studies and software tool that have been used to extract statistical measures of software quality. The research plans for this research were explained and highlighted five key steps in completing this study. The research design explains the steps to plan the hypothesis of extracting software quality metrics for the hybrid application system. Several cases studies have been used to excrete data for the proposed software quality settings of the AO/OO application. Gaps in the research were identified based on an extensive literature review. The literature review showed that many AO measurements can be identified in hybrid application systems. Common OOP and AOP measures were found with similar effects on the software measures. Static and dynamic metrics were chosen for analysis and evaluation. The effect of the AO quality of a hybrid application system was studied. The proposed AOP quality parameters for the hybrid application system have been showed. AOP extracted metrics from these projects have

examined and compared to literature review results. Several assumptions for the AOP quality measures were developed. Software quality frameworks have been proposed for the AO/OO hybrid application system based on the outcome of the case study approach and literature review analysis. The proposed framework was the subject of a review and evaluation in Chapters 5 and 6.

3.2 Research Methodology

As a research tool, a literature review is one of the major approaches to conducting research [28] [29]. A literature review can be narrowly defined as the systematic manner in which previous work is collected and produced. Proper analysis as a research tool provides a solid basis for the development of scientific and theoretical growth [30]. A literature review can tackle research questions without the force of a single study by combining observations and insights from several empirical findings. It may also help to provide insight into how research is carried out is interdisciplinary and diverse. A literature review is an excellent way to summarise research findings to explain the evidence at the metadata level. It helps to identify areas where further research is required to build theoretical frameworks and conceptual models. However, accepted approaches to explaining and presenting documentation are often underdeveloped and not consistently adopted [28]. Mitchell, Ortu, and Orrú techniques give authors a significant chance to build their investigation on the erroneous assumptions. Sometimes, when the analytical method is correct, the problems contribute significantly [31]. Different study methods are available. There are systematic, semi-systematic and integrative approaches, which depend on each type of approach to be adapted and the consistency of the very efficient implementation. A semi systematic review approach is one of the excellent strategies for research approaches, such as mapping approaches or theoretical topics and identifying knowledge gaps in the literature. In such cases, a research question requires a research approach to data collection. An integrative analysis approach may be helpful when the focus of the review. It is unnecessary to cover all papers ever published on the subject, unnecessary ect but to combine perspectives to construct new theoretical models. Several questions relate to responses to research questions of software quality metrics have been selected. For example, the questions may have to do with determining the examination method for the specific search query. It can decide what the eligibility criteria are and set limits for the analysis. It can help select the data to be extracted from the document, or in assessing what form of contribution should be made.

.

A case study approach is a research method appropriate for software engineering research because it studies modern phenomena in their ordinary context. However, what makes up a case study is

defined differently among researchers, hence the quality of the studies produced. This study, based on the method of empirical research in software engineering, also emphasises experimental research. The first methodological manual [32] [33] [34] has been published. However, the studies presented range from very ambitious ordered field studies and well done with various taxonomies are used to categorise the work [35]. Zhang and Jacobsen proposed case studies and field research (39). The case study approach is well suited to software engineering research of any kind, as the subjects of the study are current phenomena that are difficult to study in isolation. However, the case studies were criticised for being less precious, impossible to general criticised and biased researchers. This criticism can be overcome by the application of robust and systematic analytical practice, by re-examining the fact that information is not only of statistical importance [40]. This study used a semi-systematic review approach, which was used to identify knowledge gaps regarding the AO/OO quality model measures [41]. Adding more, the case study approach was used to assess a software quality framework for the AO/OO quality model based upon the ISO 9126 software quality model.

3.3 Research Design:

The research design is one of the success factors for this research. The research design of this study explains the two methods of research method to achieve goals and objectives, as shown in Figure 73.1. In this research, theoretical software quality frameworks for the AO/OO hybrid application system were first developed before carrying out empirical evaluations. Research design has five major steps. In the first step, the software measures are studied from the literatures found on the published papers and books. In the second step, the Aspect Oriented parameters and quality parameters for the hybrid AO/OO application system are studied from the literature of published articles and books. In the third step, AOP quality measurement software for the AO/OO hybrid application system is proposed and then extracted from several open source code projects developed by AO/OO technologies. In the fourth step, a software quality model framework for hybrid AO/OO application systems based on the existing ISO/IEC 9126 model was implemented. This framework includes a theoretical architecture, a member selection mechanism based on AOP proposed quality measures, and a member validation mechanism. In the last step, the quality of the AO/OO application frameworks is evaluated.

.

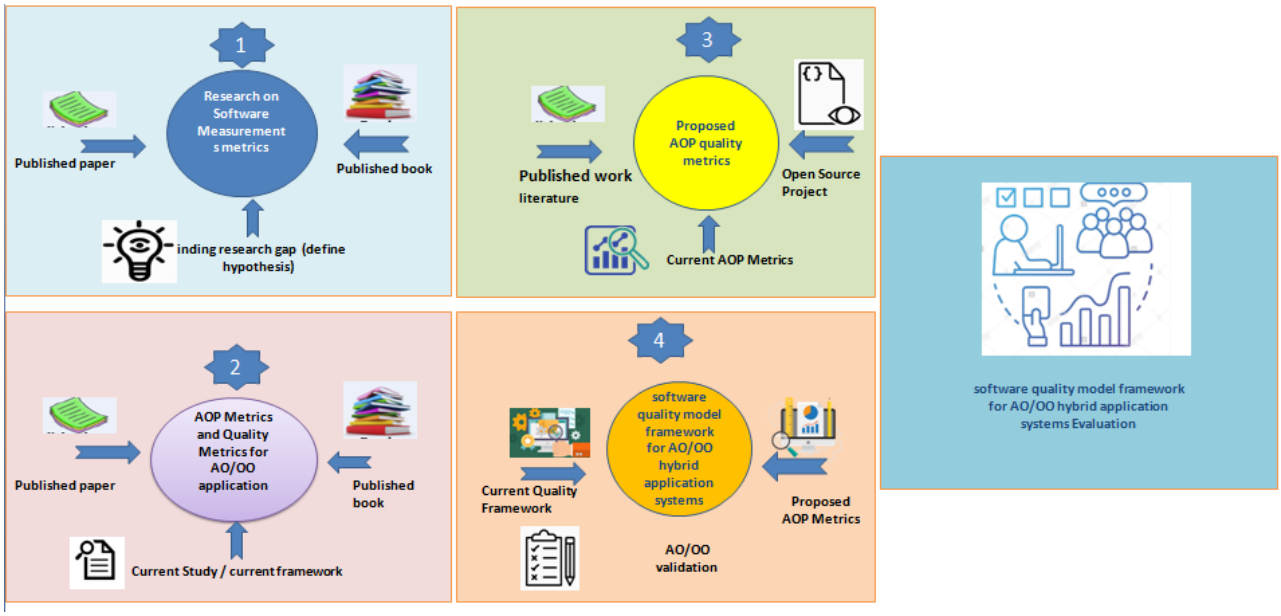


Figure 3.1 Research design steps used to carry thesis research

3.3.1 Research Design Steps details:

This section provides a brief description of all research design steps.

1. Step 1: Research of software measurements to understand AOP software measurements for hybrid application systems. A literature review research method was adopted to identify the current gap between the assessment of additional measures and the assessment of AO/OO applications. Many published papers and books were researched for software measurements and AO/OO measurements. The results of the research at this stage were discussed further.
2. Step 2: This step extends the pervious step. Many research articles and books have been researched to find the extent of AOP measurements and AOP quality measurements. The primary aim of this phase is to understand the feasibility, mechanism and scope of the new AOP measures to measure AO/OO application.
3. Step 3: Propose AOP Quality Measures This step was undertaken based on the collection of research on AOP quality measures and to find a gap in its development method. Several cases studies have been used to determine the feasibility and scalability of the proposed AOP quality measures and developed the existing AOP quality measures. Chapter 4 details the techniques used to extract the data.

4. Step 4: Software Quality Template Framework for AO/OO Hybrid Application Systems. The measures proposed in Stage 3 will feed into a new software quality model for hybrid AO/OO application systems. Furthermore, the hypotheses of this study with the response of the detailed literature review. The hypotheses analyse the results of the case studies that provided baseline elements of the proposed quality framework. This model has been proposed according to pervious quality model ISO/IEC 9126. MSE [235] was used to design the framework model and later will be assess it.
5. Step 5: Software Quality Model Framework for the Assessment of AO/OO Hybrid Application Systems. There are several frameworks to measure software quality. However, no quality measurement framework exists for the application developed by AO/OO. One of the most common ways to ensure that the framework is valid is to evaluate it and prove the proposed hypothesis and criteria for this proposed framework model. The qualitative and quantitative assessment method has been adapted AMOS and the [235] has been adapted in the assessment process. 3.3.4 Evaluation Methods

According to Cronholm, S. and Goldkuhl [253] there are several evaluation strategies that can be considered in systems evaluation strategy methods, for example “criteria based“, “goal-based” and “no-goal”. Criteria based assessment is an evaluation strategy process performed against several criteria that the system adheres to in order to be successful. Goal based assessment applies to a set of pre-defined goals in the system development. No-goal assessment is performed on system development without pre-defined objectives. Adding more, Chen, S., Osman and others [254] have been applied to three different implementation strategy methodologies “IT system used” and “IT system itself”. The IT system used strategic methods concern the involvement of users and evaluators to work on the system and collect data. The IT system itself relates more to the evaluator than to the collection of data and the conduct of the assessment. This thesis was used as a mixing method for the evaluation strategy method “goal-oriented” and “IT system”. Evaluation process is the important part of the research and to ensure that the outcomes result gets correctly and the study is done properly, an evaluation strategy has been developed for this research. Figure 3.2 explains the Evaluation Strategy Approach.

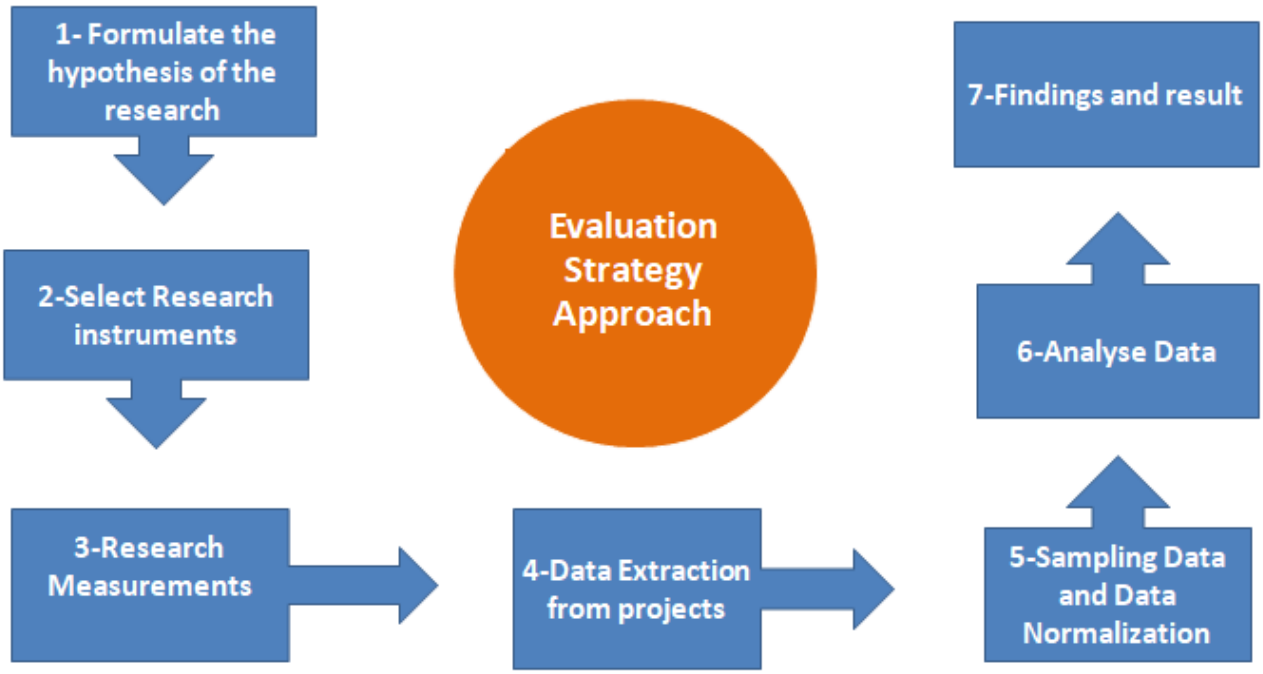


Figure 3.2 Evaluation Strategy Approach

3.3.2 Formulate the Hypothesis of the Research:

Five hypotheses of this study have been proposed based on the detailed literature review research approach, which is described in section 3.6. Five assumptions have been made regarding the new AO/OO product quality framework. These assumptions have been made based on measures were extracted from five open source projects which have been described in Section 4.2. Chapter 5 will explain these hypotheses with their evaluation. Hypotheses made are:

H1: Extensibility has improved the functionality.
H2: Complexity influences reliabilities.
H3: Service Loading Behaviour has improved the efficiency.
H4: Code weaving makes maintainability more reliable.
H5: Re usability improves the portability.

3.3.2.1 Select Research Instruments:

The case study from this study was used to gather and retrieve measurement data. Measurement tools include Code-MR, java program, algorithm that has been developed. The analysis and

evaluation of the data was done using the SPSS statistical software [238]. The first step is to extract the data, then map and normalise it, and then input it into an SPSS spreadsheet. The extraction data and all attributes will be organised to address the defined assumptions, and the evaluated; all details will be explained in Chapter5.

Research Measurements: According to Structural Equation Modelling, the extracted data will categories into directly (observable) and indirectly variables (latent). The (observable) variable can be found directly from the extracted data of the project and shows the effect of the metric values of the open source project. Standardisation techniques were used to standardise the extracted value from zero to ten. The mathematical standardisation equation was used in this process; full details will be explained in Chapter 5.

Data Extraction from Projects: Common metrics for AOP and OOP have been extracted. The common OOP and AOP measures were retrieved using the Code-MR tool and a java program. The data have been organised to meet research questions and the hypothesis; all the details will be explained in Chapter5 of the evaluation.

3.3.3 Sampling Data and Data Normalization:

In this stage, measurement results have been varied. Therefore, the extracted data has been normalised based on the size of the source code and the functionality that has been used in each project. Different development methods of AOP and OOP techniques have been used in the AO/OO project according to specifications for each project. MSE was used to model sample data for each project. The outcomes of these extracted data will fed into the product quality model, all the details will be explained in chapter5.

.

3.3.4Analyses Data:

Five differentiated projects were analysed with the help of seven statistical measures. These statistical measures ensure the data extracted is valid and reliable. These statistical measurements have been tested using SPSS tool functions.

Findings: The ultimate result of the proposed framework has been approved statistically. Full details will be explained in chapter 5.

3.4. Conducted Research experiments:

This section explains case study experiences that partially extract AO/OO measures. All the following experiments show some disadvantages of extraction of these measures.

3.4.1 Experiment 1 - Bash Script algorithm

This experiment has been conducting by creating a bash script algorithm to extract AO/OO metrics. The procedure of working the algorithm comprises these steps (Source code parser, application metadata extraction, module analyser, metrics calculator and saving the result into database) as shown in Figure 3.3 below:

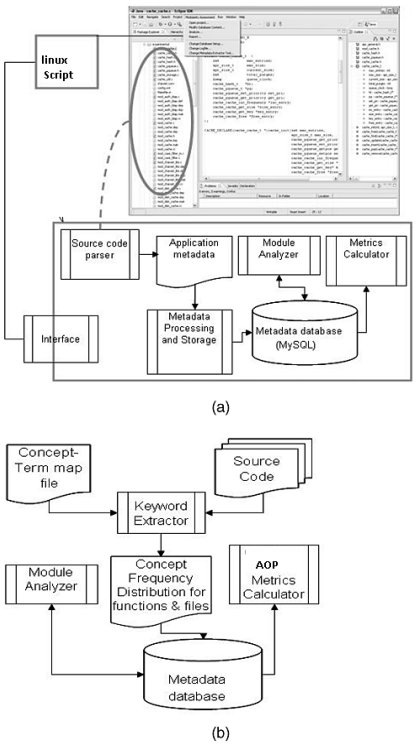


Figure 3.3 Bash script code structure

The sample code of the algorithm of finding AO/OO as it shows in Figure 3.4. The example uses several Linux command for find word pattern in the source code of hybrid application AOP/OOP techniques.

```
#!/bin/bash

umask 177  # Make sure temp files are not world readable.


TRUE=1
$AOPFILE =/var/AOPProject
#  Note that $AOPFILE must be readable
#+ (as root, chmod 644 /var/AOPProject.
TEMPFILE=temp.$$
#  Create a "unique" temp file name, using process id of the script.
#     Using 'mktemp' is an alternative.
#     For example:
#     TEMPFILE=`mktemp temp.XXXXXX`
KEYWORD=advice

ONLINE=22
USER_INTERRUPT=13
CHECK_LINES=100
#  How many lines in AOP file to check.

trap 'rm -f $TEMPFILE; exit $USER_INTERRUPT' TERM INT
#  Cleans up the temp file if script interrupted by control-c.

echo

while [ $TRUE ]  #Endless loop.
do
  tail -n $CHECK_LINES $AOPFILE> $TEMPFILE
  #  Saves last 100 lines of system AOP file as temp file.
  #  Necessary, since newer kernels generate many log messages at log on
  search=`grep $KEYWORD $TEMPFILE`
  #  Checks for presence of the "advice" phrase,
  #+ indicating a successful logon.

  if [ ! -z "$search" ] #  Quotes necessary because of possible spaces.
  then
     echo "On-line"
     rm -f $TEMPFILE    #  Clean up temp file.
     exit $ONLINE
  else
     echo -n "."        #  The -n option to echo suppresses newline,
                        #+ so you get continuous rows of dots.
  fi

  sleep 1
done

#  Note: if you change the KEYWORD variable to "Exit",
#+ this script can be used while on-line
#+ to check for an unexpected logoff.

# Exercise: Change the script, per the above note,
#           and prettify it.

exit 0


#  another alternative:


while true
  do ifconfig ppp0 | grep UP 1> /dev/null && echo "connected" && exit 0
  echo -n "."   # Prints dots (.....) until connected.
  sleep 2
done

# Problem: Hitting Control-C to terminate this process may be insufficient
#+         (Dots may keep on echoing.)
# Exercise: Fix this.



#  another alternative:

CHECK_INTERVAL=1

while ! tail -n 1 "$AOPFILE" | grep -q "$KEYWORD"
do echo -n .
   sleep $CHECK_INTERVAL
done
echo "On-line"

|
```

Figure 3.4 Algorithm procedure for extracting AO/OO metric

Another example used is the identification of many class files with the extension .java and Aspect file with the extension aj. It used the simple command file WC -L. The purposes of this test extracted all metrics using the bash script with piping and other Linux command features. There were many impediments to using a bash script. For example, the project structure, files cannot read. Linux scripts do not support analysis for all aspects and java file components. The components calling the source code do not calculate correctly and sometimes cannot count the other usage of the code in the various files of the project. Table 3.1 below shows the line of code of the selected open

source projects. The components calling the source code not calculating correctly and, sometimes, cannot count the other use of the code in different files of the project.

No.	Project Name	Java Files	Aspect file
1	Google Talk	25462	195
2	AjHotdraw	41594	2579
3	Health Watcher	8113	1827
4	AjHSQLDB	1153133	3829
5	Contract4J5	16052	1571

Table 3.1 Extracting number of java class and Aspect files

3.4.1.2 Experiment 2 -Ajatoo Application Tool

The Ajatoo tool was assessed to collect aspect measurements against design pattern metric measurements [14] [18]. It applied to three different design models implemented in OOP and AspectJ (adapter, observer, and decorator). Variety of measurements has been supported by this software, such as (attribute numbers, size, operation numbers, and weight operator per component, code lines, and tree depth. This tool has the drawback of not supported new versions of java JDK and not measure quantity of AO/OO metrics. Figure 3.5 explains it in details.

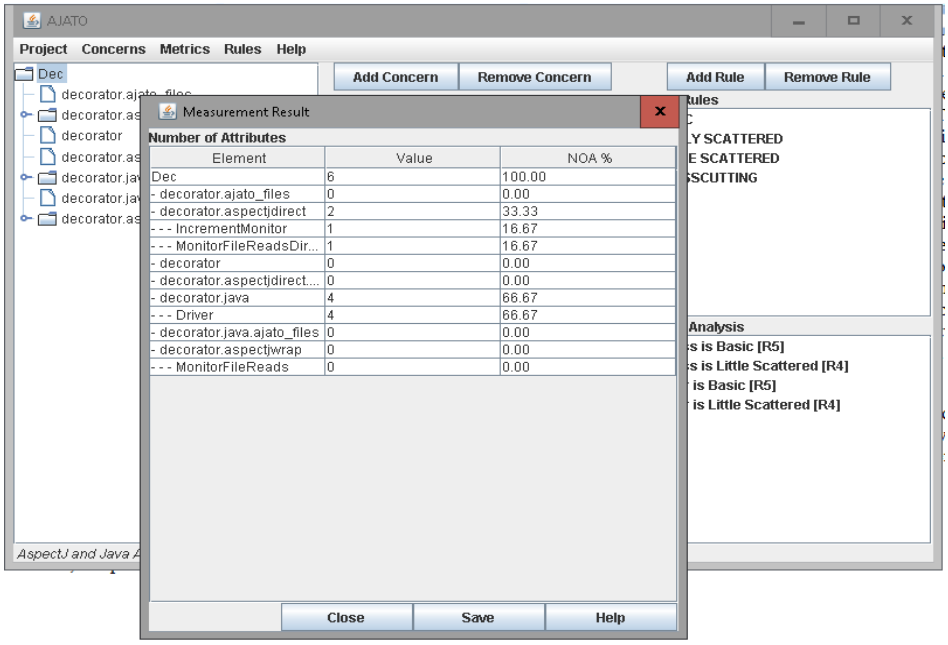


Figure 3.5 Using Ajatoo tool

3.4.1.3 Experiment 3 -AOP Metrics Application Tool

AOP Metrics are developed with the help of AspectJ. It has been tested several metrics such as WOM, LOC, DIT, LCO, NOC, CIM [14]. It also executes dependencies of units like abstraction and number of types [19]. The tool disadvantage is designed for AspectJ and Java with earlier versions and has not been revised for eight years. It will not measure aspect independently and not measure quality metrics for software [32].

3.4.1.4 Experiment 4-CMTJava Application Tool

CMTJava application tool is used for extracting complexity metrics and supports implementation of corporate standards for code complexity, testing and measuring quality metrics [33]. CMTJava measures several complexity metrics but has also a drawback that has a limited in measuring all complexity metrics.

3.4.1.5 Experiment 5 - JDepend Application Tool

The JDepend [34] application tool involved breaking down the design of the framework into many components, such as scalability, reusability and practicality. JDepend it deals with measuring the number of packages and controls it. The software design is used to generate the metrics from multiple interfaces and classes, such as abstraction, couplings, instability, primary sequence

distance, and packet dependency cycles. The tool did not support all AOP source codes and did not recognise AO metric and software quality metrics.

3.4.1.6 Experiment 6- JMetric Application Tool

JMetric is a tool to extract OO metrics. JMetric gets metrics from documentation of the source code [35]. The Tools will generate several measurements such as, number of line, number of readings and Cyclomatic complexity. Metrics are supported by a plug-in measure release of 1.3.5, under free licence of open-source module in the Eclipse IDE [36]. It determines 17 unique metrics and gives a packet dependency analyser. The draw backs of this tool are not support software quality measurements for AO technology.

3.4.1.7 Experiment 7 -Rapid miner Application Tool

Miner Rapid piloted a new design template for the application. The difficulty of using this tool is manual rule configuration and only accepts CSV pretext input files. However, it has been used to test on some hypotheses but cannot apply to all AOP measurements [37]. As is shown in Figure 3.6 below:

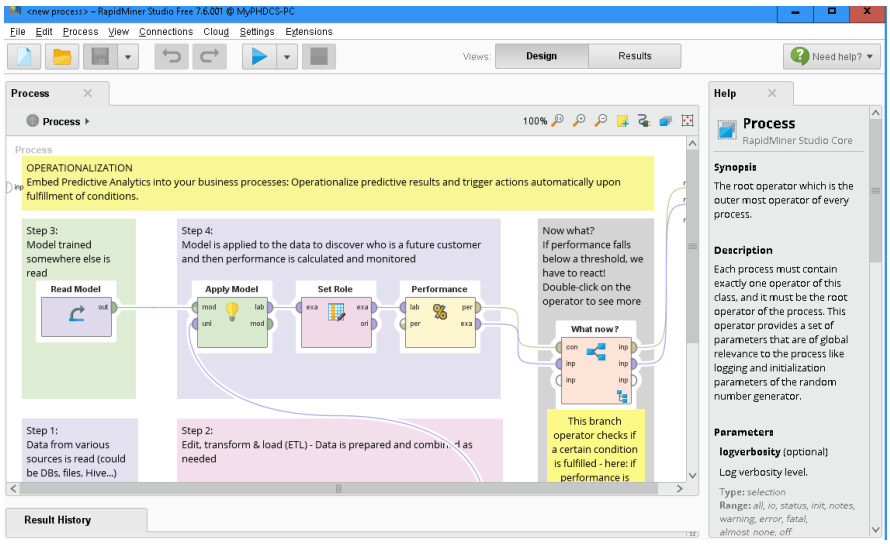


Figure 3.6 Rapid miner tool

3.4.1.8 Experiment 8 -GATE Application Tool

The GATE application tool is strong mining engineering software. GATE has an overall architecture for text engineering [38] that has been tested to find AOP source code models. AOP measures have been found in the statistical analysis. The disadvantages do not provide statistical information on each source code. Figure 3.7 shows the details of it:

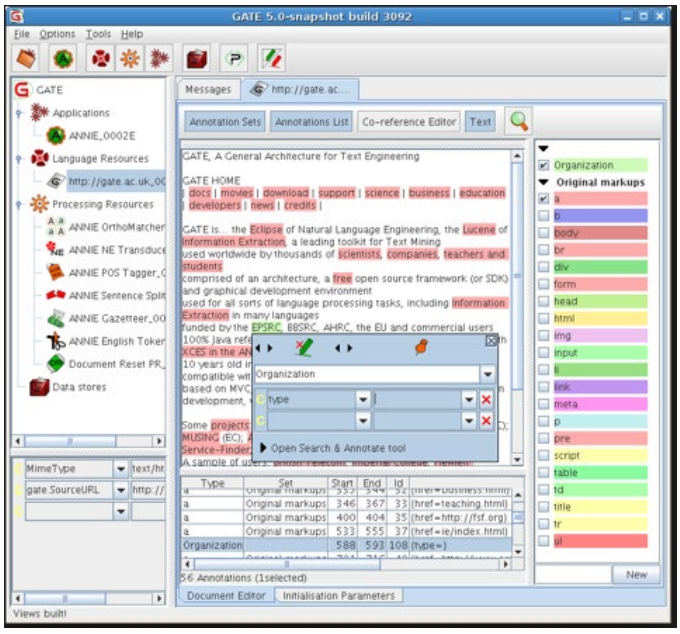


Figure 3.7 Gate text engineering tool

The above practical experimentation tools were used for extracting new AOP measurements. They were used to extracting some of the AOP measurements. There are no special tools for retrieving all AOP and OOP measurements. It is important to find tools to relate OOP to AOP measurements in order to measure the hybrid application system. All the summary of 8 experiments have been summarised in table 4. The table has been to explain the advantages and disadvantages of all these experiments. The conclusion that there are not unique tools that can support all AO/OO metrics measurements for hybrid application system. So in section 3.5, it has been used a solution for these obstacles.

Experiments for extracting AOP metrics	**Tools or application have been used**	**Results**	
		Advantages	Disadvantages
Experiment 1	Bash Script Linux	It is perfect for calculating static metrics AO/OO.	It cannot calculate dynamic metrics and reading content different file format

			of AOP.
Experiment 2	Ajatoo Application Tool [14][15]	It measures AO/OO metrics for design pattern development and generates 6 metrics. It open source application.	Limited number of metrics and limited on JDK 1.5 version and aspect old version only. It needed up to two years for upgrading.
Experiment 3	AOP Metrics Application Tool [32]	The tool has measured dependencies of units like abstraction and number of types, and measure several AOP metrics only,	It only measure AOP metrics and has limited number of measurements. Is not support measurement of AO/OO both and it is not an open source application.
Experiment 4	CMTJava Application Tool [33]	is used for extracting complexity metrics and supports implementation of corporate standards for code complexity, testing and measuring quality metrics	It is not support all quality metrics and all complexity metrics measurements.
Experiment 5	JDepend Application Tool	it deals with measuring number of packages and controls it. The software design is used to generate the metrics from multiple interfaces and classes such as, abstraction, couplings, instability, primary sequence distance, and packet dependency cycles	The tool did not support all AOP source code and did not recognize AO metric and software quality metrics
Experiment 6	JMetric Application Tool [35].	JMetric gets metrics from documentation of the source code. The Tools will generate several measurements such as, number of line, number of readings and Cyclomatic complexity.	The draw backs of this tool is not support software quality measurements for AO technology, and does not support AO/OO metrics measurements.

Experiment 7	Rapid miner Application Tool [37]	It generates metrics from design template for the applications and can customise the metrics extraction rules.	The difficulty of using this tool is manual rule configuration and only accepts CSV pr text input files, adding more cannot apply to all AOP measurements.
Experiment 8	GATE Application Tool [38]	is strong mining engineering software. GATE has an overall architecture for text engineering that has been tested to find AOP source code models. AOP measures have been found in the statistical analysis.	do not provide statistical information on each source code.

Table 3.2 Summary table for conducted research experiments3.5 Open source project developed with AO/OO source code

Five open source projects were selected for their development using AO/OO source code. The projects have been selected based on an extensive literature review [75] [101] [115] [148] [150]. These projects have been used in the case studies to extract quality parameters from the proposed AOP. These open source code project considered as hybrid AO/OO applications. The following free source code projects have been described as:

3.5.1 GTalkWAP project

GTalkWAP is a web-based application is used to access the Google chat service using WAP enable access. GTalkWAP is programmed by java and AspectJ [34]. The appearance effect is slight as far as the project has minor features and part of its implementing using crosscutting concern. The effect of an aspect is small regarding that the project has small functionalities and some of it is implanted using crosscutting concerns.

3.5.2 AJHotdraw project

This is an open source project developed with Aspect Oriented refactoring of the 2D graphic system in JHotDraw. Test AJHotDraw is a research sub-project designed to ensure behavioural maintenance between the two solutions [35]. It used java 2D graphics with object-oriented features; however, the quality of the system is varied because of the few refractory aspects of the capabilities.

The crosscutting issues were used in their functions. Based on the size of the project, it was selected for thesis experiments.

3.5.3 HealthWatcher Project

The HealthWatcher system is a feedback system designed to improve the quality of health care institutions by allowing the public to draft their health complaints. For example, complaints against restaurants and food stores, health care agencies, can intervene quickly in claims and take the action. The programs have a web-based user interface for logging complaints and carrying out several other related operations. A layered architecture and associated design tendencies [36] [37] [38] to achieve modularity and scalability. The framework was used as part of the Java and AspectJ implementation. This layered architecture enables the separation of data management, industry, communication (distribution) and presentation (user interface) concerns. This arrangement leads to a less complex code, such as when the business code intertwines with the distribution code but does not completely prevent it. For example, it is crucial to easily disentangle the transaction start and end code by using this architecture and an object-oriented language. However, in situations where it can be unravelled, a high price must be charged for this: the interfaces must be written to take care of the transaction functionality. This project has been chosen because aspect handles code distribution, competition, persistence, synchronisation and transaction.

3.5.4 AjHSQLDB Project

AJHSQLDB is the outcome of an Aspect Oriented case study on refactoring with HSQLDB which is java open source system implementing a relational database system. JHSQLDB used FEAT to search for the corresponding transversal code [39]. This is a large system because it involves logging, managing exceptions, grouping, tracing and allowing and authenticating. The systems were chosen because we believe they might answer many of our research questions.

3.5.5 Contract4J5 Project

Contract4J is free software, covered under the Eclipse-v1.0 public license. This tool used JDK of Java 5. A Contract4J5 have been developed using Java programming and Aspect Oriented Programming. Adding more, Enterprise Java and Ruby on Rails, is sponsoring Contract4J5 [39] [40]. The exception handles were used for the transversal concern in the entire project.

3.6 AO/OO metrics

The following sections explain these measures and how they were retrieved.

3.6.1 Crosscutting Degree of an Aspect (CDA)

CDA measure all modules possibly influenced by an aspect. It shows the global impact of one aspect on the other modules. Higher CDA values are preferred [7]. The WOM is another method for measuring CDA. The CDA/WOM measure may show the effect (in terms of affected modules) of an aspect. High values of CDA/WOM show that the pieces of Advice affect several modules. Aspect is explicitly or implicitly changes all modules through the intertype statement or pointcut [8]. The CDA can be considered a valid theoretical measure for both dynamic and static crosscutting [9]. The crosscutting degree of a dimension metric is used as an indicator of separation of concerns [14]. The following uses can be considered:

• High CDA values are desirable [8], as the CDA metric shows the number of modules of an Aspect and the usefulness of the Aspect. The number of explicitly named modules in the Pointcut of an Aspect must be kept low [8]. When the value of the CDA is equal to one, it required to measure inheritance or association mechanisms to separate the concerns encapsulated by the Aspect.

3.6.2 Coupling on Advice Execution (CAE)

CAE is the number of aspects containing Advice triggered by the execution of operations in a module. CAE has the second definition by [10], which counts the number of inwards arrows from Aspects to a specific module [10]. This metric is used to measure the Advice's operation; hence, a change in the Advice might affect the process [7]. A larger value of this measurement for a module means that the software module is paired with more aspects [10]. For example, if have a high CAE coupling value with low maintainability, this mean specific advice types in AOP languages reduce the quality metrics [12]. This metric depends on the Advice operations and any change in Pointcut's behaviour that might intercept it. The CAE metrics calculate the weight of the number of aspects affecting a module [7]. The information about this metrics is collected on source code, not at execution time. Adding more, CAE metric can measure Jointpoint and show the effect on the source code. CAE measures the coupling between the Aspect and the base code, but is not sensitive to the coupling rate. This measure should address many Joinpoint coupling mechanisms to measure the coupling caused by performing advice [11] [13]. According to Ceccato and Tonella [13] they claim the, CAE has several effects on source code. For instance, aspects or classes with low LOCC values and high CAE values may be used for measuring aspect interactions. The CAE/WOM metric can see how aspects affect the global behaviour of a module. The value of CAE/WOM (is directly proportional to the influence of the component aspects. The values of the CAE measure may be used as an indicator of the interaction of aspects. Therefore, they conclude that small amounts of CAE are more acceptable because the higher the CAE value, the more the class is related to the

aspects that affect it [7]. If a module has a null CAE, this means aspect has not deducted the module.

3.6.3 Concern Diffusion over Components (CDC)

The CDC records the number of classes and aspects that are primarily intended to help implement the concern [14]. Ng, Kaeli, Kojarski and others argue the CDC has the number of components that contribute to implementing the concerns and other elements that have access to them [15], whereas Kiczales, Hilsdale, Hugunin and others state that the primary purpose of the CDC is to measure classes and aspects [16]. The CDC evaluated the relationship between defects and crosscutting problems. It was found that the more a concern is spread out, the more likely there will be errors in its implementation. CDC is used to measure appearance scatter ratio in software, and these metrics are independent from the program size [17]. This metric quantifies feature scattering, considering the granularity level of components. It counts the number of classes and interfaces that support the implementation of functionality. CDC calculates the ratio of classes which deliver the implementation of a function to the total number of classes. The relative CDC represents the percentage of classes which are used to implement the feature [18]. The CDC has the number of components whose only purpose is to help with implementing a concern. In addition, it tracks the number of attribute statements, formal parameters, return types launches and local variables call their methods [9]. It can be represented as follows.

CDC= number of components that help implement a concern + number of elements that access the components that help implement the concern. This measurement can measure the extent to which a single system concern is consistent with the software design elements. If a concern directly matches the items, fewer components will be changed during the execution time.

3.6.4 Lack of Concern based Cohesion (LCC)

Concerns are defined as any property and part of the software, which can be considered a conceptual unit and treated in a modular way [23]. Concerns can range from high-level notations like security and quality of service to low-level issues such as caching and buffering. It can be functional as the rule or operational features or not functional, such as function organization and transaction management [23] [24] [25]. Cohesion is an internal quality attribute that is essential to software. A software module is consistent if it represents an abstraction of only one concept or characteristic of the problem domain. The less responsibility a module has, the more coherent it will become [24]. The measurement of cohesion is useful for assessing various aspects of software design, including the prediction of the trend to module change [22]. The LCC for a component c

includes the number of concerns assigned to it. It also counts the number of distinct concerns to which the interface is assigned, plus the number of different concerns to which the interface operations are assigned [20].

The LCC accounts for the number of concerns addressed by the evaluated component. The LCC is based on a similar measure defined within architectural levels [21]. The LCC measures the amount of cohesion for a component based on the number of concerns it addresses. A component may be a class, interface, or any other component representing a module as the implementation unit. The results are thus got for each component. In this way, it records the number of concerns mapped to each element [26]. The LCC measurement applies to each part, for example, in a class. The Aspect LCC identifies whether a component is likely to be cohesive or undetermined by measuring the number of concerns it implements. If a class contributes to implementing several concerns, it lacks cohesion. This metric looks at what matters in implementing the components and the number of responsibilities (concerns). The reasoning behind LCC is that an element that encompasses many concerns is subject to regular change. The disadvantage of the concerns resides in the requested changes during the implementation of the concern. In addition, we would expect to find a correlation between effective linkage and LCC. If a class has many concerns, it may depend on other classes because it is more involved in the overall concerns of the system [27] [28].

3.6.5 Aspect Complexity Metrics (AC)

Complexity measures for OO applications are well developed, including control of system structure, difficulty of data type and complexity of data access card [34]. AO complexity metrics can be applied for advice on aspects, number of parameters for methods in classes in OO metrics. However, Aspects also include other constructions that need to be taken into consideration, such as Pointcut, Joinpoints and Advice. Aspect-oriented complexity measures use the same object-oriented measurement techniques to measure AO components. The following sections describe certain features of the appearance complexity attribute:

Pointcut describes a condition that triggers the specific section of code to run. The AV-graph techniques were proposed by Ceccato and Tonella that may represent point cut by the AV - graph. The AV - graph represents Pointcut per node and represents the signature as entry node. The metrics measurements have calculated, including summing up the signature of Pointcut definitions

and signature complexity (call, execution, etc.). Second Techniques measure the complexity of pointcut by a regular expression. These regular phrases are given by [30]. The regular expression was represented by a string of literal symbols (e.g., as food), a keyword (such as int) and a standard character expression (such as *). The measures of the complexity of a signature refer to all functions (in the form** (*)) or specifically one (void foot (int) throws IOException) requires the same order. However, more complex signatures, like the void f*oo (int, *) throws *, are hard to analysis understand. The keywords before and around describe when the Pointcut part of the body of the Board is executed. Otherwise, the advisory structure is quite similar to that of a Java system. Inter-type instructions enable declaration of aspect priorities, custom build errors, alerts, etc. Inter-type statements allow the declaration of aspect priorities, custom compilation errors, or alerts, among others.

Advice is composed of two parts: the feature part and the Pointcut part. The Advice header consists of several parts. The name of the Advice can follow the list of standard settings, keywords before, after, or around, and return. The body of Advice does not appear is different from that of a java method. The functionality of Advice is the same as java methods. The Aspect body is contains a part of the Pointcut definition. The complexity of the Aspect body can be measure by combination of complexity of Pointcut.

The complexity of Aspect elements and classes has many similarities; both include member data and functions. The complexity of a class is the sum of the complexity of the methodologies and the characteristics of the data. Aspect complexity is classified into two categories. The difficulty of Advice is considered the Pointcut when calculating the dimension. These structures have a direct impact on the programmer's view of the code. There is several programming statements such as parent error reporting, alert and others are not considered when calculating complexity. In AspectJ, auxiliary constructor not directly impact complexity. The complexity values are summarised by the complexities of the details and aspects of the members' functions.

Cyclomatic complexity [29] depends only on the number of independent routes through the programme that can be represented by this equation (V (G) = p + 1). The inadequacy of the metric becomes clear if we understand that the Cyclomatic class ignores the nesting level of the predicate nodes. Enhancements in the form of a management structure coefficient are proposed by [30], [31] and [32]. The scope of a predicate node is a group of instructions whose performance depends on the choice created in the predicate node. The degree of nesting of the program is defined because of the variety of predicate nodes whose scope includes instruction. Therefore, the complexity of a PV program is divided into three components:

System Structure Regulation: Most systems have the same control statements, no matter which model is used. The control structure is represented by a graphic in which the nodes are orders, and the pointed edges represent the control possibilities. Nodes that have an output edge greater than one are referred to as predicate nodes.

Nesting level training nodes are used for weighting. Data type challenge: It reflects the complexity of the data used in the classes. In the control chart, the data nodes are represented as various types of nodes.

The complexity of data access represented by the relationship between the control structure and the data is represented by borders between the data nodes and the statements used. Regarding the complexity of metrics for AOP can be explained as following, the complexity of Pointcut is the summation of the complexity of the names and of Pointcut itself. Adding more, whether calling Pointcut multiple times is counted. The call comprises calling the function, and the execution comprises performing a program. Void ||int f (*)) means all called functions which have either avoided or a type of int feedback and get a setting of any kind. The concepts of OOP, AOP defines the following basic concepts, such as, Aspects include Pointcut, advice, and statements of inter-type existence. The depends is referee complexity of an AOP program is dependent on the OOP components and AOP-specific structures. Therefore, the ambiguity could be divided between the AO-specific parts of the Pointcut definitions, and the object-oriented structure classes and methods style that have used. Therefore, this study argues that AOP quality measurement should be implemented, which measures more paradigms simultaneously and supports the measurements of AO/OO software applications.

The following metrics have been compared and analysis as it have been shown in table 3.2 [104] [105] [107]. These sub-characteristics will influence these external quality attributes, such as complexity, modularity, reliability and maintainability under the software quality model of the hierarchy [104] [105] [107].

1. Coupling Advice Execution (CAE): CAE contains many aspects containing advice, triggered by the execution of operations in a module. (Complexity)
2. Crosscutting degree of an aspect. (CDA) (Modularity)
3. Lines of Class Code (LOCC) (Complexity)
4. CBO (Object Linkage): Records the number of dependencies of a class or aspect. The tools check all types used within the entire class (field declaration, method return types, variable declaration, and so on). (Modularity)

5. WMC (Weight, Method Class or Advice). It counts the number of industry statements in a class. (Complexity)
6. Depth of Inheritance Tree (DIT) (Complexity)
7. LCOM (Lack of Cohesion of Methods): Calculates metrics for the class and Advice of aspect (Modularity)
8. Quantity of returns: The number of return statements. (Complexity)
9. Quantity of loops: The number of loops (i.e., for, while, do-while enhanced for).
10. Number of comparisons: The number of comparisons (i.e. ==). (Complexity)
11. Quantity of try/catches: The number of try/catches (Complexity)
12. Number of expressions in parentheses: The number of expressions in the parenthesis. (Complexity)
13. Letter chains: The number of letter chains (for example, "John Doe").
14. Replicate channels count as many times as they appear. (Complexity)
15. Number count: Number of numbers (int, long, double, floating).
16. Several mathematical operations: The number of mathematical operations (time, split, remains, more, less, left shift, right quarter). (Complexity)
17. Number of variables: Number of variables entered. (Complexity)
18. Max nested blocks: The largest amount of nested blocks together. (Complexity)
19. Using each variable: it concern with how many times is each variable used for each method.
20. Use of all fields: it deals with how many have been used in each method. (Complexity)

No.		Complexity	Modularity	Reliability	Maintainability
1	Coupling on Advice Execution (CAE)	*	_	_	_
2	Cross-cutting degree of an aspect. (CDE)	_	*	-	_
3	Lines of Class Code (LOCC)	*		*	*
4	CBO (Coupling between objects)	_	*	_	_
5	WMC (Weight Method Class or Advice)	*		*	*
6	Depth of Inheritance Tree (DIT)	*	_	*	*
7	Coupling on Advice Execution (CAE)	_	*	_	_
8	LCOM (Lack of Cohesion of Methods)	_	*	_	_
9	Quantity of returns	*	_	*	*
10	Quantity of loops	*	_	*	*
11	Quantity of comparisons	*	_	*	*
12	Quantity of try/catches	*	_	*	*
13	Quantity of parenthesized expressions	*	_	*	*
14	String literals	*	_	*	*
15	Quantity of Number	*	_	*	*
16	Quantity of Math Operations	*	_	*	*
17	Quantity of Variables	*	_	*	*
18	Max nested blocks	*	_	*	*
19	Usage of each variable	*	_	*	*

20	Usage of each field	*	_	*	*

Table 3.2 external quality attributes for static and dynamic metrics of AOP

3.7 Proposing AOP Quality Metrics

Based on the details of the literature review in Chapter 2, Section 2.7, AOP quality measurements were proposed based on several research studies[7][8][9][10][11][12][13][15][16][17][19][20][22][23][24][25][27][28][29][30]. Five open source projects that were developed with AO/OO programming techniques were selected to extract these metrics as described in section 3.5. The AOP metrics that have been selected are: Crosscutting Degree of an Aspect (CDA), Coupling on Advice Execution (CAE), Concern Diffusion over Components (CDC), Lack of Concern-based Cohesion (LCC), and Aspect Complexity Metrics (AC). The AO metrics will affect several components of the software quality framework and the proposed software quality framework for the AO/OO. Rashid and Awais [18] have been added an extension of the ISO/IEC 9126 quality framework by Jung, Kim and Chung [50]. Table 3.3 below shows the relationships between the proposal's AOP metrics and some components of the software quality model framework. CDA and CDC measures are effects on modularity or complexity depending on the design and architecture of the software. As a result, we selected 5 deferential projects to ensure that the extracted value is accurate and reliable.

Proposing Sub-Components	**Proposing Sub-Attribute**
Extensibility	Crosscutting Degree of an Aspect (CDA)
	Joinpoints Extension (JPE)
	Concern Diffusion over Components (CDC)
	Number of Models
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	RFC Response for a Class (RFC)
	Responses for Aspect (RFA)
	Number of Aspects (NOA)
Service loading behaviour	Line Of Code (LOC)
	Number of Methods (NOM)
	Number Of Fields(NOF)
	Weighted Joint points Counts (WJC)
	Line Of Code Aspect (LOCA)
Code Weaving	Number of Children (NOC)
	Coupling Between Object Classes (CBO)
	Access to Foreign Data (ATFD)
	Coupling On Advice Execution (CAE)
Re-usability	Lack of Cohesion Of Methods (LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack of Tight Class Cohesion (LTCC)

	Lack of Concern Based Cohesion (LCBC)
	Lack of Cohesion in Joinpoints (LCJ)

Table 3.3 Proposed AOP Quality metrics

The proposed AOP metrics with components of software quality model have been categorized as it shows in Table 3.5.

Proposed AOP Quality metrics	Components of Software quality model
Crosscutting Degree of an Aspect (CDA)	Modularity or Complexity
Coupling on Advice Execution (CAE)	Coupling
Concern Diffusion over Components (CDC)	Modularity or Complexity
Lack of Concern-based Cohesion (LCC)	Cohesion
Aspect Complexity Metrics (AC)	Complexity

Table 3.5 proposed AOP metrics VS components of Software Quality model

Under the previous section 3.6, the proposed AOP quality parameters will feed the new product quality parameters for hybrid AO/OO systems. AOP quality measurements in the data extraction experience showed the relationships and effect on the product quality framework of hybrid AO/OO systems. The proposed framework is based on the output of the ISO/IEC 9126 quality model with additional metrics for measuring AO/OO quality metrics [18]. Several studies have been proposing software quality metrics, but none of them working on hybrid AO/OO systems [8] [9] [10] [11] [12] [13] [18] [50].All the steps of proposing the new framework will explain in chapter 5.

Table 3.6 below outlines the key components that may be added to the Quality Framework. The proposing product quality framework has been designed using Structure Equation Modelling. The design contains all components have multiple attribute relationships that have been developed, as shown in Figure 14 below.

Quality framework	Main Components	Proposing Sub-Components
Product Quality-Hybrid Application System	Functionality	Extensibility
	Reliability	Complexity
	Efficiency	Service loading behaviour
	Maintainability	Code Weaving
	Portability	Re-usability

Table 3.6 proposing sub-Attributes for AO/OO quality framework components

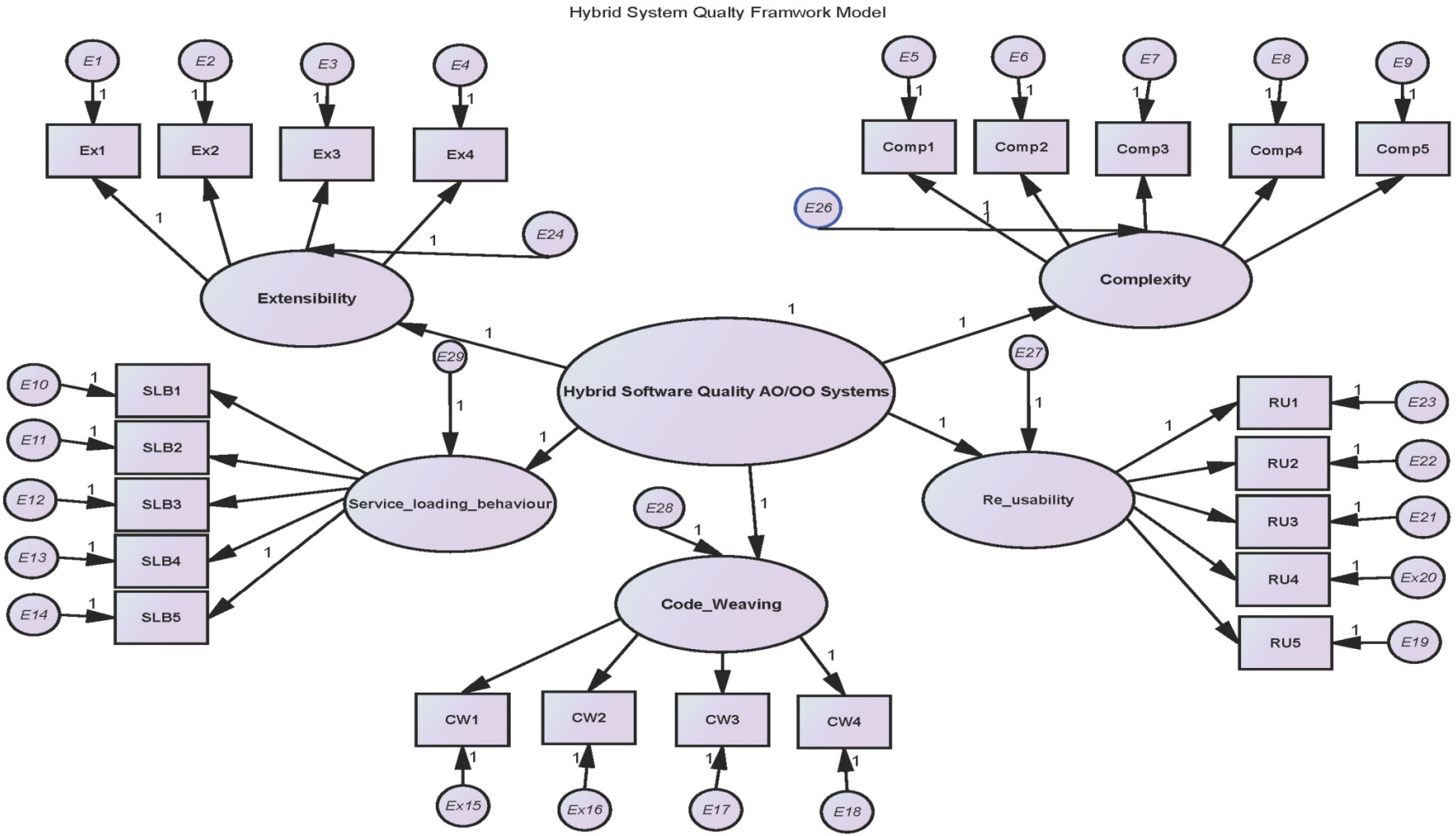


Figure 3.8 Product Hybrid AO/OO Application framework model

3.9 Conclusion

This chapter provides details on the evaluation method and process used to achieve the goal and purpose of this thesis. A comprehensive literature review and case study method were adapted for this study. The study found that eight experiments employ eight differentiation tools. The reasons for the failure of these experiments were mostly assessment tools do not support all AO techniques. Five open source projects have been selected and developed with a combination of AO/OO technology. Theoretical frameworks are proposed through an extensive empirical evaluation. This chapter has explained steps of research design, which include five steps. The evaluation process comprises seven steps aimed at ensuring the reliability and coherence of the data extracted. The framework was also designed and assessed using the structural equation model and the SPSS AMOS statistical evaluation of the proposed framework.

CHAPTER 4
DATA EXTRACTION

4.1 Introduction

The purpose of this chapter is to explain how data has been extracted from five selected projects. AOP quality metrics will be extracted based on detailed literature review and case study experiences. The chosen projects use all AO/OO development techniques. Projects vary in size, and development techniques to prove theoretical hypotheses. Data extraction techniques were carried out in two phases, the first being the extraction of common AO/OO metrics. A second step consists in categorizing each common parameter for the proposed quality parameters for the AO/OO framework. Lastly, all extracted data were organized and normalized according to a standardized mathematical equation..

4.2 Code-MR data extraction tools for Hybrid AO/OO metrics*:*

The Code-MR Software Quality Tool is used to collect general quality metrics, as shown in the graphic details below and in the table below. Code-MR is a tool used to measure software quality for projects developed in Java or C++. These tools measure static metrics that help companies produce better software. There are many plug-INS available on IDE platforms, like Eclipse and IntelliJ IDEA. The initial analysis of this study contains extracting the OOP parameters. The AOP metrics will be got by another program developed in Java to show the AOP metrics and the effects of hybrid app projects. Code-MR supports Chidamber and Kemerer metric retrieval. Other aspect-oriented programming metrics are got using the Java program and a bash script; however, the Code-MR pulls out the current AOP and OOP metrics.

4.2 Open Source Project AO/OO Hybrid System

Five open source projects were selected for their development using AO/OO source code. The projects have been selected based on an extensive literature review [75] [101] [115] [148][150] [151]. As it is explains in section 3.5. The projects are:

1. GtalkWAP
2. AJHotdraw
3. HealthWatcher
4. AjHSQLDB
5. Contract4J

Code-MR has been used to read all projects, files and folders. It has been generates several types of metrics. The study has been focused on quality metrics for common metrics of AO/OO technologies. These results have been combined with extracted AOP quality metrics and unified in quality metrics for these hybrid application systems. Adding more it normalised and categories each project in one table as it shows in the following sections. The threshold values in this plug in are unified customisable it have been unified all threshold values between 0 and 1 values and later unified in on table. This means 1 is the maximum value and 0 is the minimum.

4.3.1 GTalkWAP project AOP Quality Metrics extraction

Code-MR Plug-in was used as an open-source tool to retrieve OOP metrics. AOP metric it was getting using the shell script and a special program developed by the java language. Code-MR has partially got AOP metrics, for example, complexity, size, coupling, and cohesion. Code-MR plug-in has been graphical views of all the details of the OOP metrics, as shown in Figures 4.1 to 4.4. The green colour means that this metric has low values; light green means low-medium, orange colour, high values and red colour. Figure 4.1 shows the complexity, coupling, lack of cohesiveness and size values. Size is 47% low-medium, and the complexity is 18% is low-medium, and the rest are low. The other two measures have smaller values, and the tool has the right quality measures. The downsides are not to read the aspect file extension, but to recognise the folder and templates affected by the Java files.

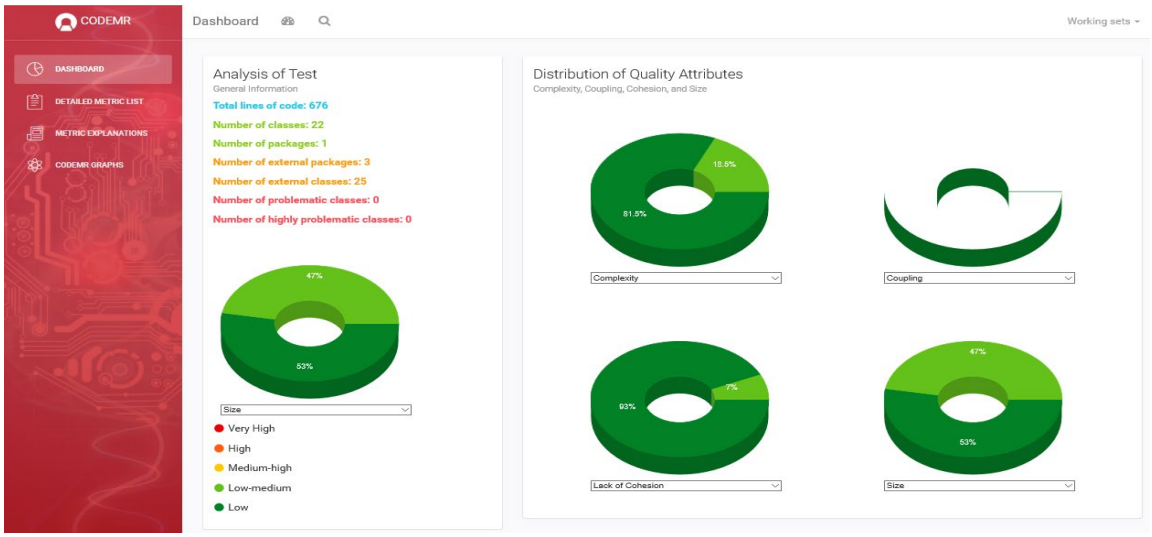


Figure 4.1 General Information about Metrics for Google Talk project

Figure 4.2 shows all metric values using a bar graph and shows the relationships between all packages. It shows the packet size represented by the size of the circle, and the colours represent the lack of cohesion and coupling, as shown in the node representation graphs.

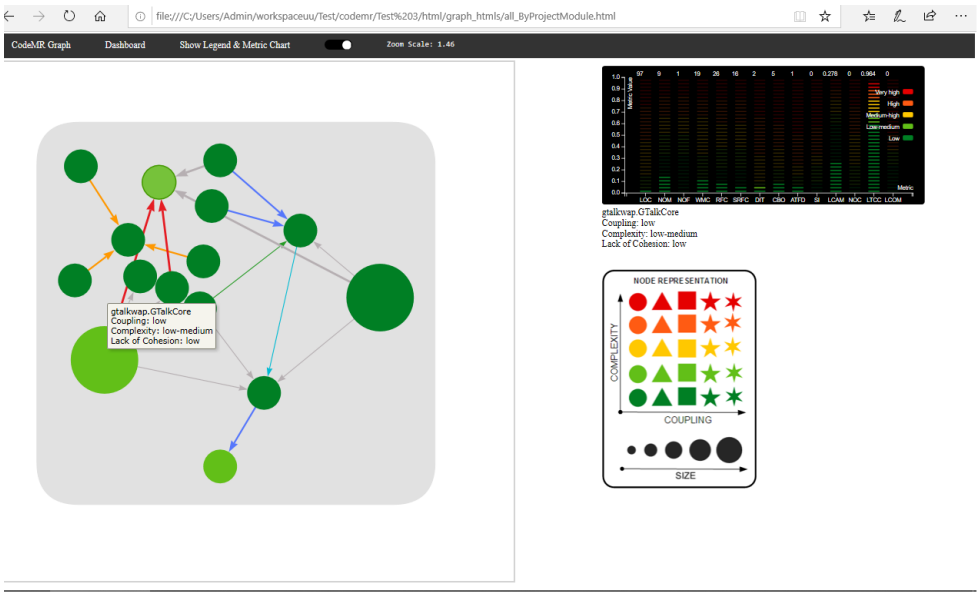


Figure 4 coupling and Cohesion values regarding packages Gtalk Project

The Code-MR plug-in shows how the modules relate to each other. The measurements have different colours, from the bottom to the top (dark green, pale green, yellow, orange and red), as shown in Figure 4.3. Colours get warmer while complexity of an entity increases. Because red colour is associated with danger in most cultures, red colour shows high values for all metrics. While the shape shows that the more corner of shape means more values of the metrics and the size of the shape shows the bigger effect of specific metrics.

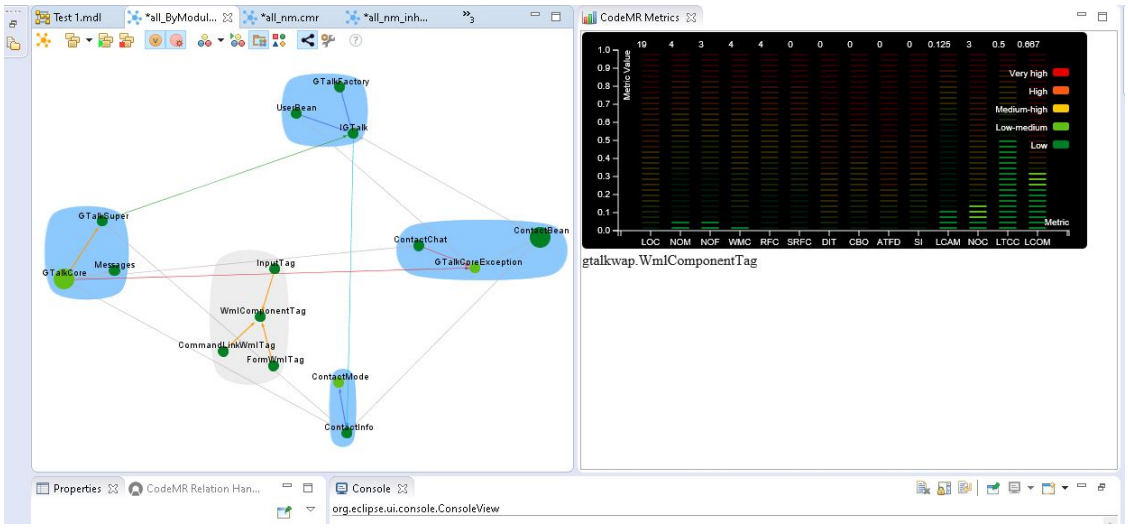


Figure 5 Metrics, values regarding Modularity Gtalk Project

Figure 4.4 shows all the details of metric values for each Java class in the project regarding showing module metrics. The results show that the project values have normal values in the project because it shows a normal distribution of each model in the packages and classes. The effects display all classes except the aspect. aj files. The tool will read all the project files and packages and then generates varies types of metrics with different techniques.

List of all classes (#22)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	GTalkCore					97	low-medium	low	low	low-medium
2	ContactMode					18	low-medium	low	low	low
3	GTalkCoreException					10	low-medium	low	low	low
4	CommandLinkWmlRen...					81	low	low	low	low-medium
5	Tags					76	low	low	low	low-medium
6	ContactBean					64	low	low	low	low-medium
7	ContactInfo					47	low	low	low-medium	low
8	UserBean					43	low	low	low	low
9	RosterRenderer					42	low	low	low	low
10	GTalkSuper					33	low	low	low	low
11	InputRenderer					27	low	low	low	low
12	InputTag					24	low	low	low	low
13	ContactChat					21	low	low	low	low
14	DataRosterTag					20	low	low	low	low
15	WmlComponentTag					19	low	low	low	low
16	IGTalk					14	low	low	low	low
17	CommandLinkWmlTag					12	low	low	low	low
18	ActionMethodBinding					11	low	low	low	low
19	Messages					10	low	low	low	low
20	FormWmlRenderer					9	low	low	low	low
21	FormWmlTag					5	low	low	low	low
22	GTalkFactory					4	low	low	low	low

Figure 6 classes Metrics Details Gtalk Project

This project has twenty-four java files and four aspect files which have extension .java and .aj respectively. Section 3 mentioned how extracting the metrics for this project using a bash script. This project considers being a small project because it contains a twenty-four class file and four aspect files. Line of code metrics shows the share of aspect is 0.07% from the entire Line of Code as it shows in table 4.1. The portion of the source code of Aspect is small. However, it's a major impact of CDC and CDA on modularity metrics and other metrics, as explained in table 4.2.

The extracted metrics has been categorized on subcategories as it appears in the table 4.1 below:

Main components of metrics	**Sub- components of metrics**
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	Response For a Class (RFC)
	Specialization Index (SI)
Coupling	Number of Children (NOC)
	Coupling Between Object Classes (CBC)
	Access to Foreign Data (ATFD)
Cohesion	Lack of Cohesion of Methods(LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack Of Tight Class Cohesion (LTCC)
Size	Line Of Code (LOC)
	Number of Fields (NOF)
	Number of Static Methods (NOSM)
	Number of Methods (NOM)
	Number of Static Fields (NOSF)
	Number of Overridden Methods (NORM)

Table 4.1 OOP Metrics and some AOP Metrics categorization GTalkWAP project

$$\boldsymbol{EQ4.1} \ldots\ldots\ldots\ldots\ldots \qquad \boldsymbol{Nor(x)} = \frac{(\boldsymbol{x} - \mathbf{min}\ (\boldsymbol{x}))}{\mathbf{max}(\boldsymbol{x}) - \mathbf{min}(\boldsymbol{x})}$$

The Normalization Equation 1 has normalized the values between 0 and 1. The equation explains below:

Main Components of Metrics	Sub- Components of metrics	Total values	Normalized values Nor(x)
Complexity	WMC	11.318	0.259
	DIT	0.271	
	RFC	0.697	
	SI	48.146	
Coupling	NOC	0.181	0.0
	CBC	0.090	
	ATFD	0	
Cohesion	LCOM	0.225	0.01
	LCAM	0.157	
	LTCC	0.315	
Size	LOC	42.875	0.12
	NOF	1.136	
	NOSM	0.090	
	NOM	3.409	
	NOSF	0.636	
	NORM	0	

Table 4.2 OOP Metrics and some AOP Metrics using Code-MR plug-in for GTalkWAP project

4.3.2 AOP quality metrics extraction using java written program:

- **Crosscutting Degree of an Aspect (CDA) metrics**: In Gtalk project it consists of has two aspects; each of them contains eight Pointcuts, which affects 40 methods in 4 modules.

  The equation of calculating the CDA metrics as following:

  EQ4.2..... CDA = Total of Effected Methods / Total of the Pointcuts

  = 40/8 = 5

  The value of CDA metrics is five, which indicates the high values of CDA. Another indication for finding the effect of CDA metrics on modules according to this equation (CDA/WMC =5/6.5), which is equal to 0.76 and indicates the Advice effect of several modules.

- **Coupling on Advice Execution (CAE) Metrics**:

  In the project Gtalk, it has a 0 JoinPoint defines in Aspects. CAE metrics is shows coupling effect on the modules when the Aspects are executing. The second indication is showing the effect on method execution by using this formula (CAE/WOM).

  EQ4.3.... CAE/WOM =0/6.5, which is equal to 0, which means there are high calling methods. Still, there is no joint point and Advice defined in the Aspect to demonstrate the effect of coupling on the modules.

- **Concern Diffusion over Components (CDC) metrics:**

  The CDC metrics has been calculated using the following formula: CDC = number of components that assist in implementing a concern + number of components that access the components. The value of CDC of this project is six, which means is have high values of CDC among the current project. Another indication of measuring CDC metrics effects using this following formula

  EQ4.4..... CDC Effect = CDC values /Total number of Classes and Aspect

  Therefore, there are 24 java classes and four aspect files; if we take an average affect overall the project, this will be

  CDC Effect = CDC values /Total number of Classes and Aspect

  =6/28

  =0.214

  The value of 0.214 is considered as average values regarding the size of the project.

- **Lack of Concern-based Cohesion (LCC) metrics:**

There are two nun functional concerns of Lack of Concern-based Cohesion (LCC) in this project; therefore, the value of LCC is equal to 2. These projects have high LCC values; this sign means that the project has a cohesive solution in concerns implementation.

- **The Aspect complexity (AC) metrics**

The Aspect complexity (AC) of is counting number of Pointcuts, method calls, signature definition, wildcard and (||) operator. This project contains 8 Pointcuts 4 of them having the complexity of 3 because we have count one for Pointcut and one for the wildcard one for calling the function (3*4). The other four Pointcuts are more complex. AC metrics will be more complex because it contains wildcard in their signature and || operator, which counted as four for each calling specific function (4*4). The total complexities of this project are equal to 26 after normalising it. AC metrics value is equal to 0.351 which shows complex projects regarding the size of the entire project. It is important to normalise the extracted metrics data; it has been normalised the values between 0 and 1. The normalisation formula can be found below:

$$EQ4.5 \ldots\ldots \boldsymbol{Nor(x)} = \frac{(\boldsymbol{x}-\mathbf{min}\ (\boldsymbol{x}))}{\mathbf{max}(\boldsymbol{x})-\mathbf{min}\ (\boldsymbol{x})}$$

Table 4.3 explains all the extracted metrics between 0 and 1. The maximum values are size, which is equal to 1. AOP measures for complexity are 0.351, showing a complex project compared to the size of the AOP. The project has low values of coupling; this means the magnificent structure of this project. The low cohesion of this project showed that it favours high readability and maintainability. Modularity values are 0.1, showing that there is a deep connection between the system modules. Modularity metrics values show the number of modules independent of any other module in the system.

.

	Size						Complexity				Coupling			Cohesion		Modularity	
No.	Project Name	Number of Class files	Number of Aspects files	Line of code aspect	Number Pointcuts	No. Of Aspects	WMC: Weighted methods per class	DIT: Depth of Inheritance Tree	RFC: Response for a Class	Cyclomatic Complexity of Aspect CCA	CBO: Coupling between object classes	NOC: Number of Children	Coupling on Advice Execution (CAE)	LCOM: Lack of cohesion in methods	Lack of Concern-based Cohesion (LCC)	Concern Diffusion over Components (CDC)	Crosscutting Degree of an Aspect (CDA)
1	GGTalk	224	44	69.928	88	22	6.5	1.7	3.9	26	0.09	0.18	00	0.22	02	66	55
Total Values for each factor	107.928						38.1				0.27			2.22		11	
Nor(x)	1						0.351				0.0			0.02		0.10	

Table 4.3 Bash Script & java program AOP metrics Extraction for GtalkWAP project

4.3.3 Discussion

The metrics values of (AO/OO) have extracted using two techniques: Code-MR tools and developed java program. Tables 10 and 11 have been combined in table 4.4. Table 4.4 below shows the average values for all AO/OO metrics out of the combined values of the average performance.

	Size	Complexity	Coupling	Cohesion	Modularity
Average Values	0.56	0.30	0.0	0.015	0.10

Table 4.4 OOP/AOP metrics values for Gtalk project

In Table 4.4, the average AO/OO metrics values have been clarified. Both tables present similar results for all parameters. The metric scale is equal to 0.56, or 50%. These findings suggest Gtalk project is a complex project. Total values for complexity measurements are 30%. These results showed that the OOP source code has medium complexity, which will reduce maintenance and testing costs. The coupling values are tiny, and the coupling for AO/OO metrics is higher. These results show that low coupling implies an additional separation of unbound source code. As the cohesion value was high, so part of the source code at a certain time in the source code is connected. The modularity values are 0.10, and explain a good connection between the related modules. Modularity metric values show how many modules in the system are independent from another module

4.4.11 AJHotdraw project

The metrics extraction of OOP for AJHotdraw project has been explained in Figure 4.5 to 4.8. Green colour of the metrics means that this measure has low values; light green colour of the metrics means low-medium, orange colour of the metrics means high values and red colour of the metrics means very high.

In AjHotdraw project, it displays a standard distribution of most packages and classes. Figure 4.5 shows the overview of the class that has high cohesion. The circles represent class size and relationships among the classes.

.

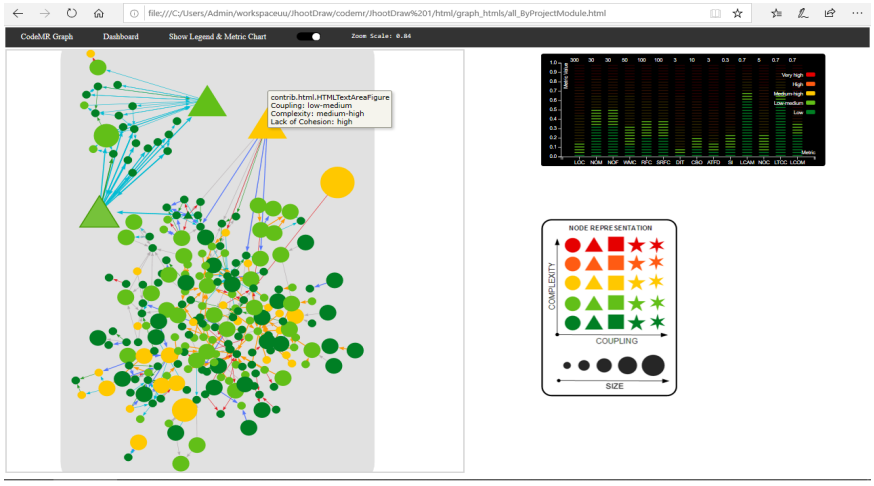


Figure 4.5 AjHotdraw project metrics

Figure 20 shows the general quality metric distribution represented by pie charts. Figure 4.6 shows the ration of complexities metrics equal to% 20.9, which shows medium-high. The Cohesion metrics equal to % 19, which show high values, and the value of 12.6% of the size metrics shows that it is a high values.

.

Figure 4.6 General Quality Metrics Distribution AjHotdraw project

Figure 4.7 shows Cohesion value is large in this project and it needs to be improved in some of its packages to reduce the high values.

Figure 4.7 Quality Metrics by Packages AjHotdraw project

The AjHotdraw project has a large complexity value, with 360 external entities as it shows in Figure 4.8.

Figure 4.8 Complexity metrics in sub chart AjHotdraw project

In this project, AjHotdraw has 291 java files and 31 aspects files with extension. Java and. aj. The percentage of the line of code is small. However, it has a significant impact from CDA and CDC on modularity measures, as it appears in Table 4.5. The metrics were classified in the measurements taken from the subcategories, as shown in Table 4.6.

Main Components of Metrics	Sub- Components of metrics	Total values	Normalized values Nor(x)
Complexity	WMC	22.274	0.33
	DIT	1.532	
	RFC	11.913	
	SI	0.172	
Coupling	NOC	0.419	0
	CBC	0.852	
	ATFD	0.017	
Cohesion	LCOM	0.229	0
	LCAM	0.345	
	LTCC	0.358	

Size	LOC	97.245	1
	NOF	1.142	
	NOSM	0.216	
	NOM	6.497	
	NOSF	0.593	
	NORM	0.324	

Table 4.2 The metrics extracted values from AjHotdraw project

Main components of metrics	**Sub- components of metrics**
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	Response For a Class (RFC)
	Specialization Index (SI)
Coupling	Number of Children (NOC)
	Coupling Between Object Classes (CBC)
	Access to Foreign Data (ATFD)
Cohesion	Lack of Cohesion of Methods(LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack Of Tight Class Cohesion (LTCC)
Size	Line Of Code (LOC)
	Number of Fields (NOF)
	Number of Static Methods (NOSM)
	Number of Methods (NOM)
	Number of Static Fields (NOSF)
	Number of Overridden Methods (NORM)

Table 4.6 OOP Metrics and some AOP Metrics categorization AjHotdraw project

All the values of the metrics have been normalized as it shows in Table 4.7.

4.4.2. AOP quality metrics extraction using java written program:

- **Crosscutting Degree of an Aspect (CDA) metrics**:

  In an AjHotdraw project, number of Aspects is 53, number of Pointcuts 37 that have called 180 methods, so formula of calculating

  EQ4.6 CDA metrics is CDA = Total of Effected Methods / Total of the Pointcuts

  =180/37

  =4.86 Another indication of finding the effect of CDA metrics on modules by using this formula (CDA/WMC) CDA/WMC =4.86/22.7, CDA/WMC =4.86/22.7, This is equal to 0.21 and indicates the Advice effect of several modules. AjHotdraw shows high values of the CDA, which indicate many modules affected by aspects of Pointcut and intertype declaration.

- **Coupling on Advice Execution (CAE) metrics**: in the AjHotdraw project the number of Advice is equal to 12. CAE is calculating number of Advice that executes JoinPoint and the type of execution of the method by using the keyword before, after, or around in the Advice. These results show that there are nine joint points. Another indication of finding CAE metrics effect on method by using the formula EQ 4.7..... CAE= (CAE/WOM)

  CAE/WOM =12/22.7, which is equal to 0.52, which means high calling methods which effect calling 9 Joinpoints in 12 Advice defined in aspect. These results have explained the effect of coupling on the base code for project modules.

- **Concern Diffusion over Components (CDC) metric**: In the project AjHotdraw, we can calculate the CDC metrics by the following formula

  EQ 4.8 .....CDC = number of components that help implement a concern + number of components that access the components.

  The CDC's value is 150, which means it has high CDC values among the project; there are 291 java classes and 31 aspect files. If we take an average affect overall the project (AO/OO), this can be showed the average values of CDC metrics for the AO/OO programming techniques:

  EQ 4.9 .....CDC Effect = CDC values /Total number of Classes and Aspect

  =150/322

  =0.46

- **Lack of Concern-based Cohesion (LCC) metrics:**

  LCC Metrics is calculated by counting all Concerns. The total number of the LCC metrics for all the packages:

LCC metrics value is equal to five for the Commands package. These concerns are three non-functional and two concerns with an interface responsible for adding and removing GUI commands.

LCC metrics value equal to five for Figure package which are functional Concerns to read and write on Figure attributes and adding other functionalities.

LCC value of the Aspect in AjHotDraw project is equal to ten.LCC metrics value for this project is high values, which is means the project has cohesive aspect concerns implementation

- **Aspect Complexity of (AC) Metric**s: The Aspect complexity (AC) of is counting number of Pointcuts, method calls, signature definition, wildcard and || operator. The project AjHotdraw has 37 Pointcuts.

  The values of AC metrics of this project as following:

  An AC metrics value of package figures is equal to 46.

  AC metrics values of package Undo is equal 10.

  An AC metrics value of package commands is equal to 235.

  AC complexity measurements involved calculating Pointcuts and intertype declaration and Advice. As it mansions in the section above, the Advice can be measured the same as methods and contains control statements and iteration, making it complicated. The Pointcut is more involved in all other packages because they have a very complex signature with wild cards, ||, &&, methods calls. The total values of AC are 282, which mean this project has high AC metrics values. Table 4.8 below shows all the details of AOP metrics for project AjHotdraw. The normalisation range values are from 0 to 1. AOP metrics for complexity AC is equal to 0.634, which shows the high value, which is considered as a complex project. The project has low values of coupling 0.006; this means the magnificent structure of this project. The average cohesion values of this project showed the project has less supporting high readability and maintainability. The modularity values are 0.298, which shows the system has a good connection between modules in the system. Modularity metrics show how many modules have independent of another module in the system.

.

	Size						Complexity				Coupling			Cohesion		Modularity	
No.	Project Name	Number of Class files	Number of Aspects files	Line of code aspect	Number Pointcuts	No. Of Aspects	WMC: Weighted methods per class	DIT: Depth of Inheritance Tree	RFC: Response for a Class	Cyclomatic Complexity of Aspect CCA (Advice, Pointcuts, JoinPoint)	CBO: Coupling between object classes	NOC: Number of Children	Coupling on Advice Execution (CAE)	LCOM: Lack of cohesion in methods	Lack of Concern-based Cohesion (LCC)	Concern Diffusion over Components (CDC)	Crosscutting Degree of an Aspect (CDA)
2	AjHotdraw	291	31	83.193	37	53	22.27	1.53	11.93	282	0.85	0.4	12	0.23	10	150	4.86
Total Values	495.193						317.73				13.25			10.23		154.86	
Nor (x)	1						0.634				0.006			0		0.298	

Table 4.7 Code-MR & java program AOP metrics Extraction for AjHotdraw project

4.4.3 Discussion

The metrics values of (AO/OO) have extracted using two techniques Code-MR tools and developed java program. Tables 4.6 and 4.7 have contained the details results for AO and OO separately. The combination of these tests showed roughly the same outcome. Table 4.8 displays the combination values.

	Size	Complexity	Coupling	Cohesion	Modularity
Average Values	1	0.48	0.002	0.0	0.29

Table 4.8 OOP/AOP metrics values for AjHotdraw project

In Table 4.9, is showing average values for AO/OO metrics for all metrics parameters. Tables (4.7, 4.8) have similar results. The metrics size is equivalent to 1, which represents the maximum values of the findings. These results suggest AjHotdraw is a complex project in terms of the size of the source code. The total values of complexity measures are 0.48, which is high regarding other metrics. High complexity leads to more cost in maintenance and testing. The coupling values are very low, which 0.02 for AO/ OO metrics. These findings suggest that low coupling involves additional separation of independent source code. High cohesion values mean that parts of the source code at any point in the code are not connected. However, the cohesion was lower than the coupling in this project, which was not recommended. The modularity values are 0.29, this shows that a deep link between the modules of the project and the source code. Modularity metrics values are equal to 0.29 shows the number of modules in the system with one module independent of another. The AjHotdraw has average modularity metrics. Overall, the total values of the AO/OO metrics parameters values are acceptable except the size of the project is high. The results show the AjHotdraw project is positively affecting quality.

4.5.1 Health Watcher Project:

The Code-MR has been to extract part of AOP metrics, for example, complexity, size, coupling, and cohesion. Figure 4.9 below present's measures taken from a Healthwatcher hybrid application project developed by AOP and OOP. Colour green means that this metric has low values; Light green means low-middle, the colour orange means high values, and the colour red means very high. The value of Complexity metrics is 8.9% is orange, which is shows low value. Coupling metrics value is very low, almost zero, which involves additional separation of independent source code. The cohesion value is 4.3%. This means that it is difficult to discover what code related to the module and cannot jump between modules and keep track of the source code of the project. The value size of the project is equal to 63%, which is considered as a high value.

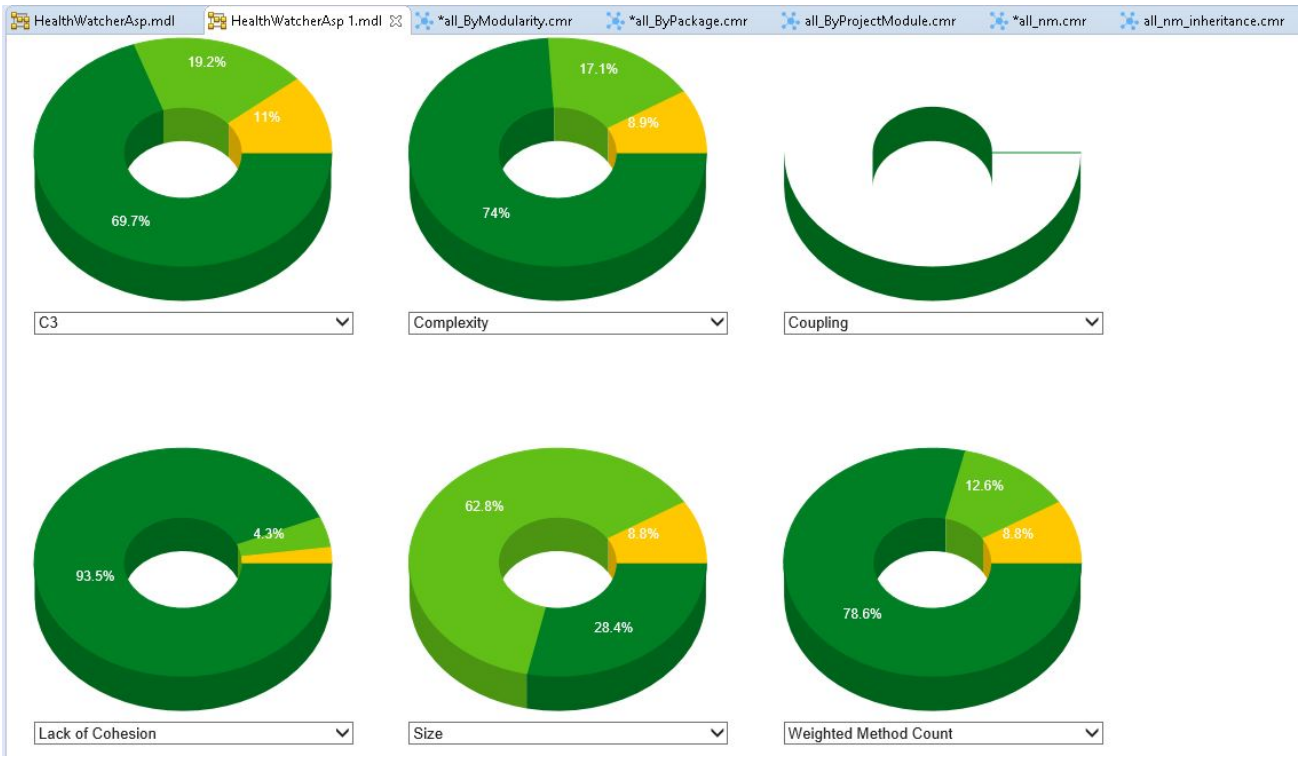


Figure 7 all metrics of Health watcher project

Figure 4.10 shows the average values for all metrics; pie metrics chart is shows each package with specific metric values. The picture below is shows the numbers of entities for each part of source

code. The distribution of the source code is normally distributed and almost all metrics have an accepted value.

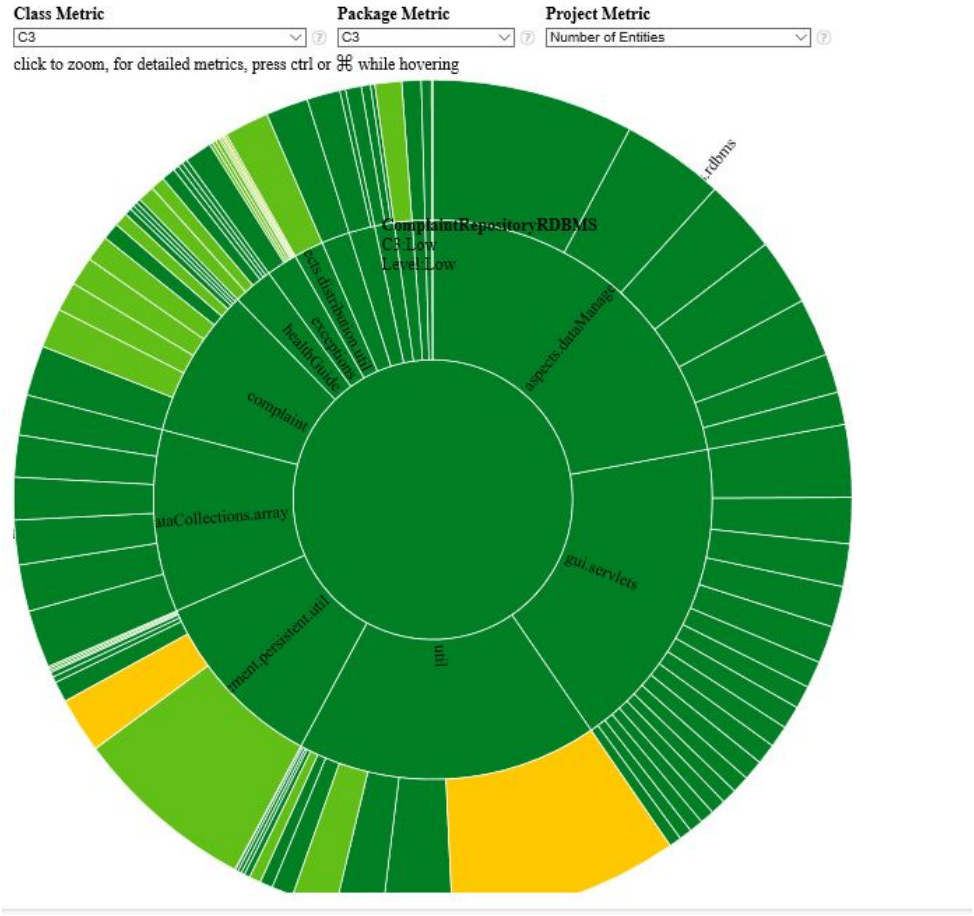


Figure4.10 metrics values by project packages

The relationships between module and packages have been explained in Figure 4.11. Colour green is shows that strong relationships between classes of the packages in the project

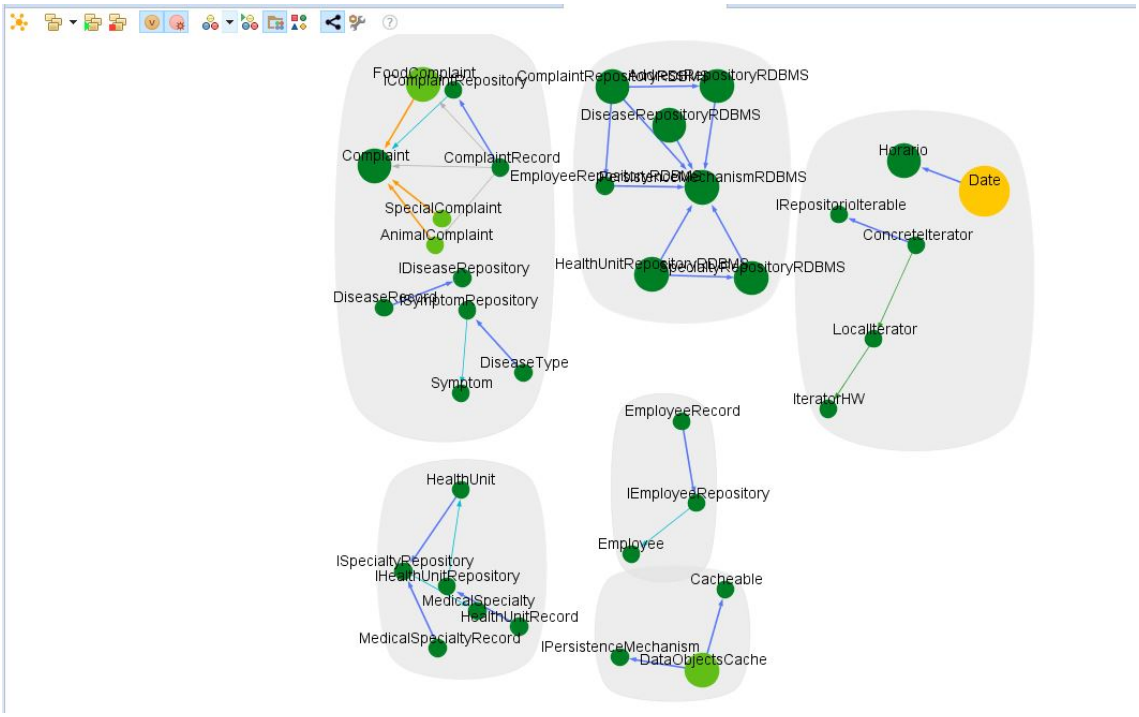


Figure 4.11 packages metrics distribution in the Health watcher Project

The module distributions of the project have been extracted as it shows in figure 4.12. There are several modules having good relationships and the values of the metrics are acceptable for HealthWatcher project.

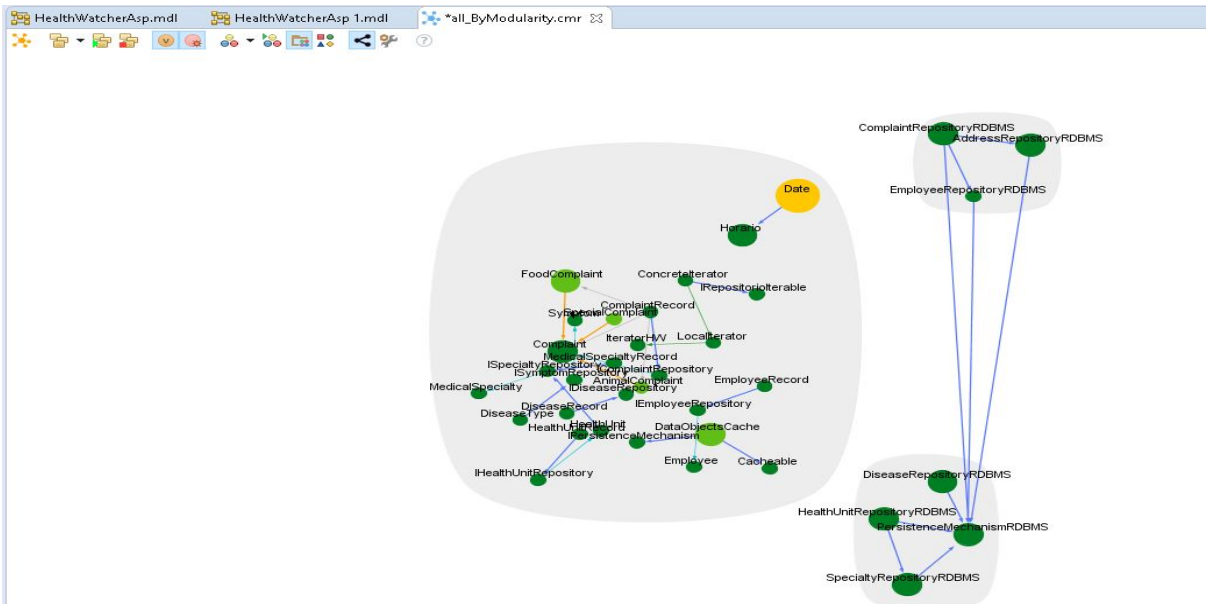


Figure 8 Module distributions for metrics for Health watcher project

This project has 90 java files and 41 aspects file with extension .java and .aj consequently. This project considers a medium project size regarding aspect file and java files; it contains a 90 class file and 41 aspect files. For instance, Line of code metrics shows the percentage of aspect is 0.622%

from the complete Line of Code, as shown in table 18 below. The portion of the Line of Code of Aspect has almost % 80 percentage of the entire Line of Code. However, it has a significant impact on modularity metrics and other metrics, as explained in table 4.10. All the metrics values have been normalised using specific equation and the range will be from 0 to 1. The normalisation equation explains below:

$$\boldsymbol{EQ\ 4.10} \ldots\ldots \boldsymbol{Nor(x) = \frac{(x - \min(x))}{\max(x) - \min(x)}}$$

The metrics has been categorized the extracted metrics on subcategories as it appears in the table 4.9 below:

Main components of metrics	**Sub- components of metrics**
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	Response For a Class (RFC)
	Specialization Index (SI)
Coupling	Number of Children (NOC)
	Coupling Between Object Classes (CBC)
	Access to Foreign Data (ATFD)
Cohesion	Lack of Cohesion of Methods(LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack Of Tight Class Cohesion (LTCC)
Size	Line Of Code (LOC)
	Number of Fields (NOF)
	Number of Static Methods (NOSM)
	Number of Methods (NOM)
	Number of Static Fields (NOSF)
	Number of Overridden Methods (NORM)

Table 4.9 OOP Metrics and some AOP Metrics categorization HealthWatcher project

Main Components of Metrics	**Sub- Components of metrics**	**Total values**	**Normalized values Nor(x)**
Complexity	WMC	9.058	0.187
	DIT	1.022	

	RFC	4.540	
	SI	0.0008	
Coupling	NOC	0.057	0
	CBC	0.333	
	ATFD	0.0459	
Cohesion	LCOM	0.224	0.002
	LCAM	0.205	
	LTCC	0.206	
Size	LOC	70.529	1
	NOF	1.218	
	NOSM	0.712	
	NOM	2.839	
	NOSF	0.632	
	NORM	0.0114	

Table 4.10 OOP Metrics and some AOP Metrics using Code-MR plug-in for Health Watcher project

4.5.2 AOP quality metrics extraction using java written program:

- **Crosscutting Degree of an Aspect (CDA) metrics:**

  The equations of CDA metrics as following

  EQ 4.11..... CDA = Total of Effected Methods / Total of the Pointcuts

  The outcomes results show that the number of aspects is 56, number of Pointcuts 109 that have called 375 methods, so the CDA is

  CDA = Total of Effected Methods / Total of the Pointcuts

  =375/109

  = 3.44

  Adding more CDA has an effect on methods execution is indicate by calculating this formula (CDA/WOM).

  CDA/WOM =3.44/9.05,

  This is equal to 0.38 and indicates the Advice effect of several modules.

  HealthWatch project shows the average values of the CDA, which indicate many modules affected by Pointcut and intertype declaration.

- **Coupling on Advice Execution (CAE) metrics**: The CAE metrics can be calculating using the following formula

  EQ 4.12..... CAE= a number of components that help implement a concern + number of components that access the components.

  The value of the CAE is equal to 200, which means it has outstanding high values of CDC among the project, because there are 90 java classes and 41 aspect files. If we take an average to affect overall the project, the outcomes of the result will be as followings:

  EQ 4.13..... CAE = CDC values /Total number of Classes and Aspect

  =200/131

  =1.52

- **Concern Diffusion over Components (CDC) metrics:**

  The CDC Metrics are calculated by counting the number of concerns. For instance, it can be calculated: The Concurrency-Control package has four functional concerns. The Data-Management package has 12 functional concerns of reading, writing, and synchronisation techniques. The Distribution package has two concerns for data transaction management methods. The Exception handler package contains one concern of exception handling functionality.

- **Lack of Concern-based Cohesion (LCC) metrics:**

  LCC Metrics is calculated by counting all Concerns. The total number of the LCC metrics for all the packages are explained below:

  The Concurrency-Control package has four non-functional concerns of synchronisation implementation. The Package Data-Management has 12 functional concerns for data processing of implementing reading, writing, and synchronisation techniques. The package distribution contains two concerns of data transaction management methods. The package Exception-Handler provides one concern with exception handling functionality. The total number of LCC metrics is 19, which shows this project has high values of LCC

- **Aspect complexity of (AC) metrics:** The Aspect complexity (AC) of is counting number of Pointcuts, method calls, signature definition, wildcard and || operator. The calculation will be as followings:
  The project HealthWatcher has 109 Pointcuts.
  The package Code-Enforcement has multiple Pointcuts, which has 5 AC metrics values, the number represents the signature of Pointcut, wild cards, &&, and various functions calls.
  The package Concurrency-Control has 40 AC metrics values; these values include aspect and Pointcuts control statements, signature, function calls, and "||," "&&" operator. The package Data-Management has 125 AC Metrics values.
  The package Distribution has 60 CCA metrics values.
  The package Exception-Handling has 150 CCA Metrics values.
  The package Update-State has 20 CCA metrics values.
  So the total values of CCA metrics are 400. The values of Aspect Complexity Metrics are high regarding the number of Pointcuts and Aspect for this project.

Table 4.11 are showing all details of AOP metrics for project HealthWatcher. The metrics values have been normalised and the range of values from 0 to 1. The maximum value is size, which is equal to 1. AOP metrics value for complexity is equal to 1, which shows the high value; therefore, the project is considered a complex project. The project has low values of coupling 0; this means the magnificent structure of this project. The small cohesion values of this project showed the project has less supporting high readability and maintainability. The modularity values are 0.476, which shows the system has an excellent connection between modules in the system and there are some modules have independent of another module in the system.

	Size						Complexity				Coupling			Cohesion		Modularity	
No.	Project Name	Number of Class files	Number of Aspects files	Line of code aspect	Number Pointcuts	No. Of Aspects	WMC: Weighted methods per class	DIT: Depth of Inheritance Tree	RFC: Response for a Class	Cyclomatic Complexity of Aspect CCA (Advice, Pointcuts, JoinPoint)	CBO: Coupling between object classes	NOC: Number of Children	Coupling on Advice Execution (CAE)	LCOM: Lack of cohesion in methods	Lack of Concern-based Cohesion (LCC)	Concern Diffusion over Components (CDC)	Crosscutting Degree of an Aspect (CDA)
3	Health Watcher	90	41	44.560	109	56	9.05	1.02	4.54	400	0.333	0.05	11	0.22	19	200	3.44
Total Values for each factor	340.56						414.61				11.383			19.22		203.44	
Nor (x)	0.816						Size				0			0.019		0.476	

Table 4.11 Bash Script & java program AOP metrics Extraction for Health Watcher project

4.5.3 Discussion

The metrics values of (AO/OO) have extracted using two techniques Code-MR tools and developed java program. As a result, Tables 18 and 19 provide some results. The combined tests gave approximately the same result. Table 20 below shows the combined average efficiency values.

	Size	Complexity	Coupling	Cohesion	Modularity
Average Values	0.93	0.59	0.0	0.01	0.4

Table 4.13 OOP/AOP metrics values for Health Watcher project

In Table 4.13, the average values for OO/AO metrics have been clarified with all parameters in both tables (4.11, 4.12) which have similar results. The metric size is equivalent to 0.93, which represents a high value. The overall values of the complexity parameters are 0.59. These results showed the source code for OO and AO presents an average complexity in this project. With less complexity, maintenance and testing costs are reduced. Coupling values are very low, and the cohesion of AO/OO measures is higher. Cohesion value was high, so part of the source code at a certain time in the source code is connected. The modularity values are 0.40, suggesting a strong link between the source codes. Modular metric values show how many modules in the system are independent of another module

.

4.6.1 AjHSQLDB Project

The Code-MR has been to extract part of AOP metrics, for example, complexity, size, coupling, and cohesion. Code-MR plug-in graphically displays all OOP metric details, as shown in Figures 4.13 to 4.16. The green colour represents the small values of the metric. Light green represents low to medium values, orange colour means high values, and red colour means very high values. The AjHSQLDB project has many issues regarding quality attributes. They are Tree Map representation for difference quality attributes in the system, such as Complexity, line of code and size. Filters apply to each measure; for instance, the filter applies to the complexity of Figure 4.13. Complexity values are high, so most packages have high values of complexity metrics and represented by red colour.

.

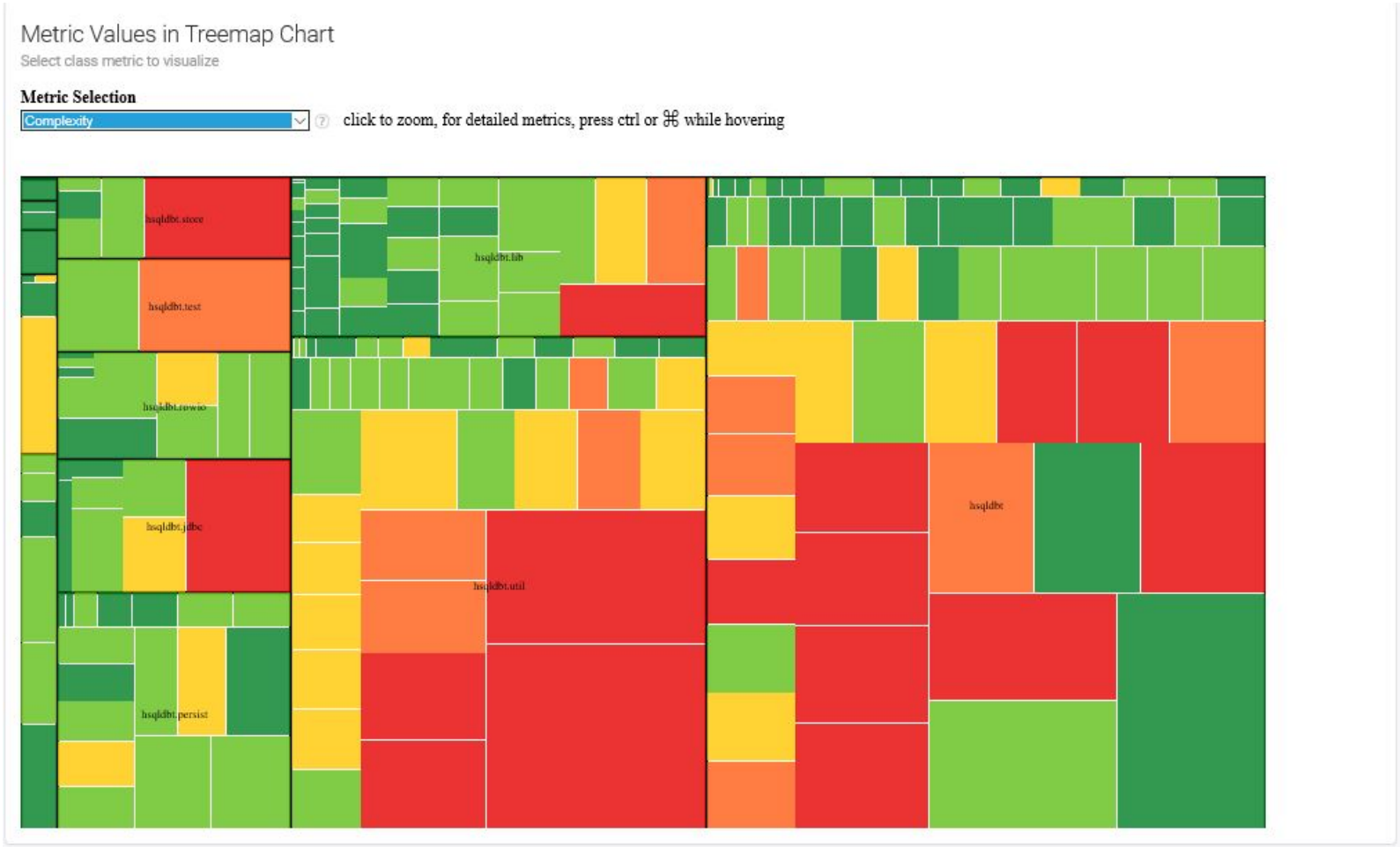


Figure 9 Metrics Values in Tree Map Chart name of the packages in complexity metrics values in AjHSQLDB Project

The relationships between project components packages and classes have been a display in figure 4.14. There are some packages are complex that have shown in red colour. AjHSQLDB project shows unique quality attributes with difference geometric shapes. Complexity of some of the AjHSQLDB project packages is high, as represented by the red colour. The coupling forms are shown as squares or triangles. The coupling patterns show that each class with high coupling has more corners that show more points of interaction with the other classes

.

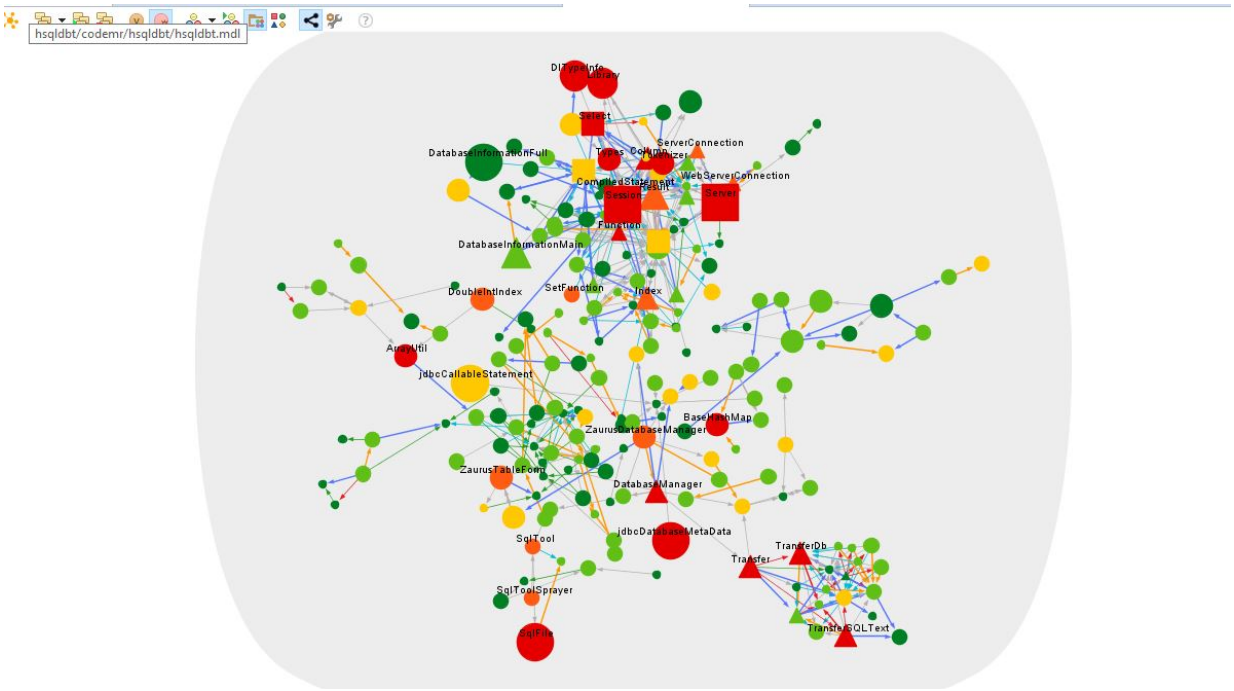


Figure 4.14 all package details with the metrics values AjHSQLDB Project

The interaction between modules has been display in Figures 4.15. High interaction between modules indicates intensive calling methods, classes and aspects, as it display by high number of lines between modules.

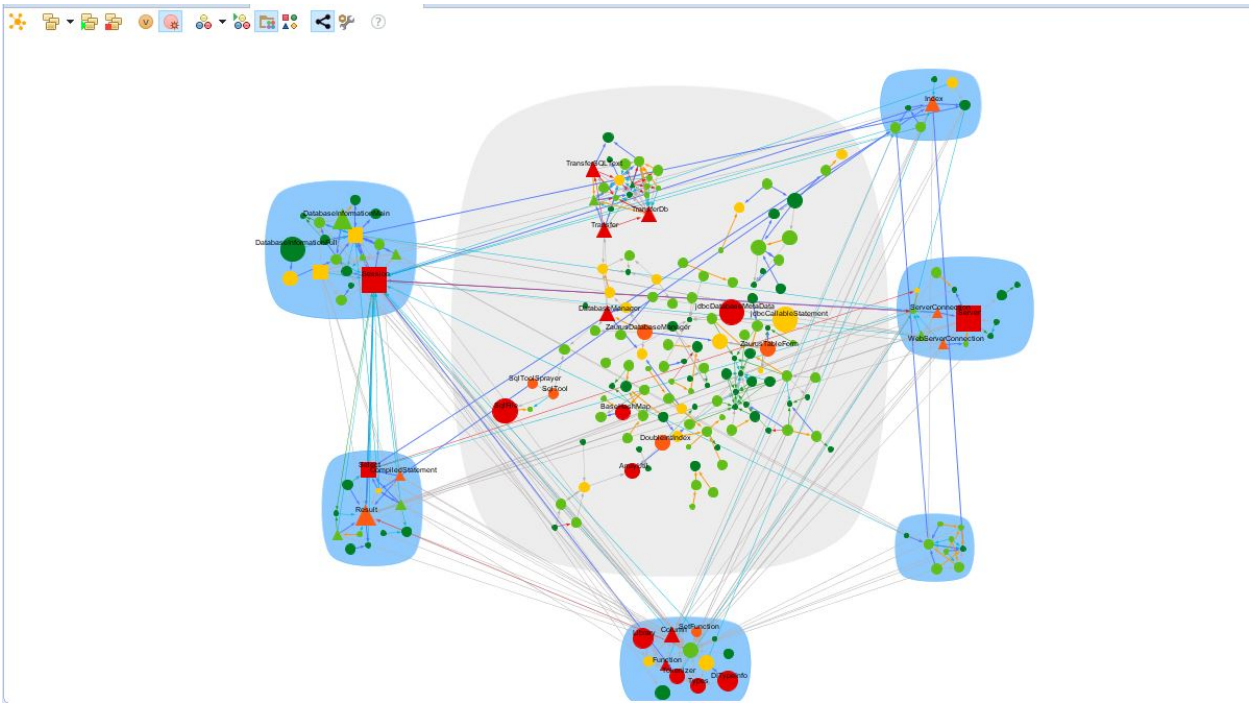


Figure 4.15 AjHSQLDB Project Quality metrics distribution by modules

The quality attributes for the AjHSQLDB project can be found in Figure 4.16. Packages are shown in bubble shapes. There are capabilities of showing different analysis. The analysis can be conducted at three levels: project, package, and class. Figure 4.16 shows lack of cohesion is high in the util package, and it is less in separate packages with red circle colour. All other measurements have been collected and analysis for other metrics.

.

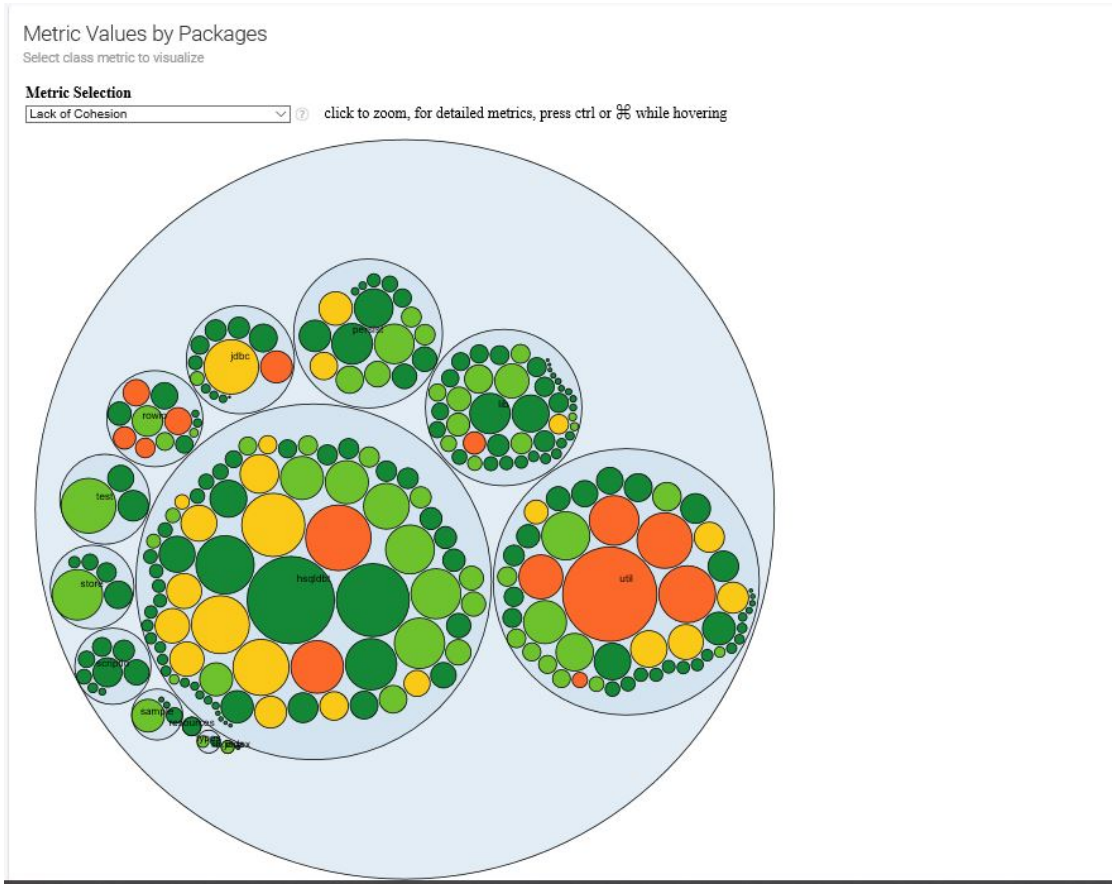


Figure 4.16 Metrics value by packages AjHSQLDB Project

In the AjHSQLDB project, the quality attributes show that 36 classes have problem classes with numerous lines and more complexity. Figure 4.17 shows numerous problems of complexity, lack of cohesion, and the size with the red colour means that it has high values.

Figure 4.17 General Quality Attribute for AjHSQLDB Project

The metrics components have been categorized into subcategories as it appears in the table 4.13 below:

Main components of metrics	**Sub- components of metrics**
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	Response For a Class (RFC)
	Specialization Index (SI)
Coupling	Number of Children (NOC)
	Coupling Between Object Classes (CBC)
	Access to Foreign Data (ATFD)
Cohesion	Lack of Cohesion of Methods(LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack Of Tight Class Cohesion (LTCC)
Size	Line Of Code (LOC)
	Number of Fields (NOF)
	Number of Static Methods (NOSM)
	Number of Methods (NOM)
	Number of Static Fields (NOSF)
	Number of Overridden Methods (NORM)

Table 4.13 OOP Metrics and some AOP Metrics categorization AjHSQLDB project

This project AjHSQLDB has 275 java files, and 25 aspects files with extension .aj. Line of Code measurements indicate that the percentage of aspect is 0.5% from the complete code line, as shown in Table 4.14. However, it has a significant impact from CDA and CDC on modularity measures and other measures, as discussed in Table 4.15.

The Normalization Equation has normalized the values between 0 and 1. The equation explains below:

$$\boldsymbol{EQ\ 4.14}\ \boldsymbol{Nor(x)} = \frac{(\boldsymbol{x}-\mathbf{min}\ (\boldsymbol{x}))}{\mathbf{max}(\boldsymbol{x})-\mathbf{min}\ (\boldsymbol{x})}$$

Main Components of Metrics	Sub- Components of metrics	Total values	Normalized values Nor(x)
Complexity	WMC	72.32	0.441
	DIT	1.419	
	RFC	35.063	
	SI	40.149	
Coupling	NOC	0.360	0.003
	CBC	1.826	
	ATFD	0.161	
Cohesion	LCOM	0.468	0
	LCAM	0.461	
	LTCC	0.377	
Size	LOC	306.192	1
	NOF	4.237	
	NOSM	7.216	
	NOM	10.466	
	NOSF	7.216	
	NORM	0.555085	

Table 4.14 OOP Metrics and some AOP Metrics using Code-MR plug-in for AjHSQLDB project

4.6.2 AOP Quality Metrics Extraction Using Java Written Program:

- Crosscutting Degree of an Aspect (CDA) metrics:
  The equations of CDA metrics as following
  EQ 4.15..... CDA = Total of Effected Methods / Total of the Pointcuts
  In an AjHSQLDB project, the total number of aspects is 50, the total number of Pointcuts 68 that have called 205 methods.
  CDA = Total of Effected Methods / Total of Pointcuts
  =205/68
  =3.01
  Another indication is calculating by using the following formula (CDA/WOM)
  CDA/WOM =3.01/72.32,
  = 0.041.
  This indicates the Advice effect on several modules. AjHSQLDB project shows the average values of the CDA, which indicate many modules affected by Aspects of Pointcut and intertype declaration.

- **Coupling on Advice Execution (CAE) metrics:** The CAE metrics can be calculating using the following formula
  EQ 4.16 .....CAE= a number of components that help implement a concern + number of components that access the components.
  The number of Advice is equal to 105; therefore, the CAE metrics are equal to 105. The second sign is showing the effect on method execution by using this formula (CAE/WOM).
  CAE/WMC =105/72.32, which is like 1.45 and shows several modules' Advice effect. The AjHSQLDB project shows high CAE values, which means many modules are affected by Advice's execution

.

- **Concern Diffusion over Components (CDC) metrics:** The CDC metrics has been calculated using the following formula:
  CDC = number of components that assist in implementing a concern + number of components that access the components.
  The value of CDC of this project is 13, which means is have low amounts of CDC among the project; there are 275 java classes and 25 aspect files if we take an average effect overall the project it can be found by the following formula:
  EQ 4.17..... CDC Effect = CDC values /Total number of Classes and Aspect

=13/300

=0.04

- **Lack of Concern-based Cohesion (LCC) metrics**:

  The LCC metrics values of project AjHSQLDB are five. LCC counts several types of concerns, such as non-functionalities and functional features of method implementation. This project has law values of LCC metrics, so the AjHSQLDB is not a cohesive project.

- Aspect complexity (AC) metrics:

  Aspect complexity (AC) of is counting number of Pointcuts, method calls, signature definition, wildcard and || operator. Calculation will be as followings. The project has 68 Pointcuts. The project has 160 AC Metrics values, including summing up of complex Pointcuts with function calls, before and after keyword with || and && operators. Adding more, it counts the number of Aspects, intertype declaration and control statement, loop statement, and methods calls. Table 4.16 below is showing all details of AOP metrics for project Health Watcher. Normalisation values range from 0 to 1. The maximum values are Size, which is equal to 1. AOP metrics for complexity are equal to 0.455, showing the medium value, which is considered not a complex project. The project has low values of coupling 0.179; this means the good structure of this project. The small cohesion values of this project showed the project had not supported high readability and maintainability. The modularity values are 0.018, which shows the system has a less impact on the connection between other modules in the system. It shows how many modules have independent of another module in the system.

	Size						Complexity				Coupling			Cohesion		Modularity	
No.	Project Name	Number of Class files	Number of Aspects files	Line of code aspect	Number Pointcuts	No. Of Aspects	WMC: Weighted methods per class	DIT: Depth of Inheritance Tree	RFC: Response for a Class	Cyclomatic Complexity of Aspect CCA (Advice, Pointcuts, JoinPoint)	CBO: Coupling between object classes	NOC: Number of Children	Coupling on Advice Execution (CAE)	LCOM: Lack of cohesion in methods	Lack of Concern-based Cohesion (LCC)	Concern Diffusion over Components (CDC)	Crosscutting Degree of an Aspect (CDA)
4	AjHSQLDB	275	25	153.16	68	50	72.32	1.41	35.06	160	1.82	0.36	105	0.46	5	13	3.01
Total Values for each factor	571.16						268.79				107.18			5.46		16.01	
Nor(x)	1						0.455				0.179			0		0.018	

Table 4.15 Bash Script & java program AOP metrics Extraction for AjHSQLDB project

4.6.3 Discussion

The metrics values of (AO/OO) have extracted using two techniques: Code-MR tools and developed java program. As a result, Tables 4.15 and 4.16 provide some results. The combined tests gave approximately the same result. Table 4.17 below is displays the combination values of average performance.

	Size	Complexity	Coupling	Cohesion	Modularity
Average Values	1	0.45	0.1	0.0	0.018

Table 4.16 OOP/AOP metrics values for AjHSQLDB project

The average values of AO/OO between various parameters have been clarified in table 4.17. All metrics values in table 4.15 and 4.16 have similar results. The metric size value is equal to 1, which represents the maximum values for the metrics. These findings show the AjHSQLDB is a complex project. Total values for complexity parameters equal 0.45. These results showed that the OOP and AOP source code have medium complexity values. For instance, less complexity, maintenance and testing costs are reduced. The coupling values are 0.1, and the coupling of the OOP/AOP measures is higher than the cohesion measures. These results suggest that high coupling which means less separation of unrelated source code. Small values of cohesion values mean that a portion of the source code at a point in the code base is not connected. Modularity is 0.018, values of modularity measures show that fewer system modules are independent of another module.

.

4.7.1 Contract4J5 Project

Code-MR has been to extract part of AOP metrics, for example, complexity, size, coupling, and cohesion. The Code-MR plug-in was graphical, shows that the functionality of all OOP measurement details, as shown in figures 4.18 to 4.21. The green colour represents the small values of the metric. Light green represents low to medium values, orange represents high values, and red shows that the code for accessing on-demand data cannot be reached. In the Contract4J5 project, some packages have a big size in several classes as it appears in Figure 4.18.

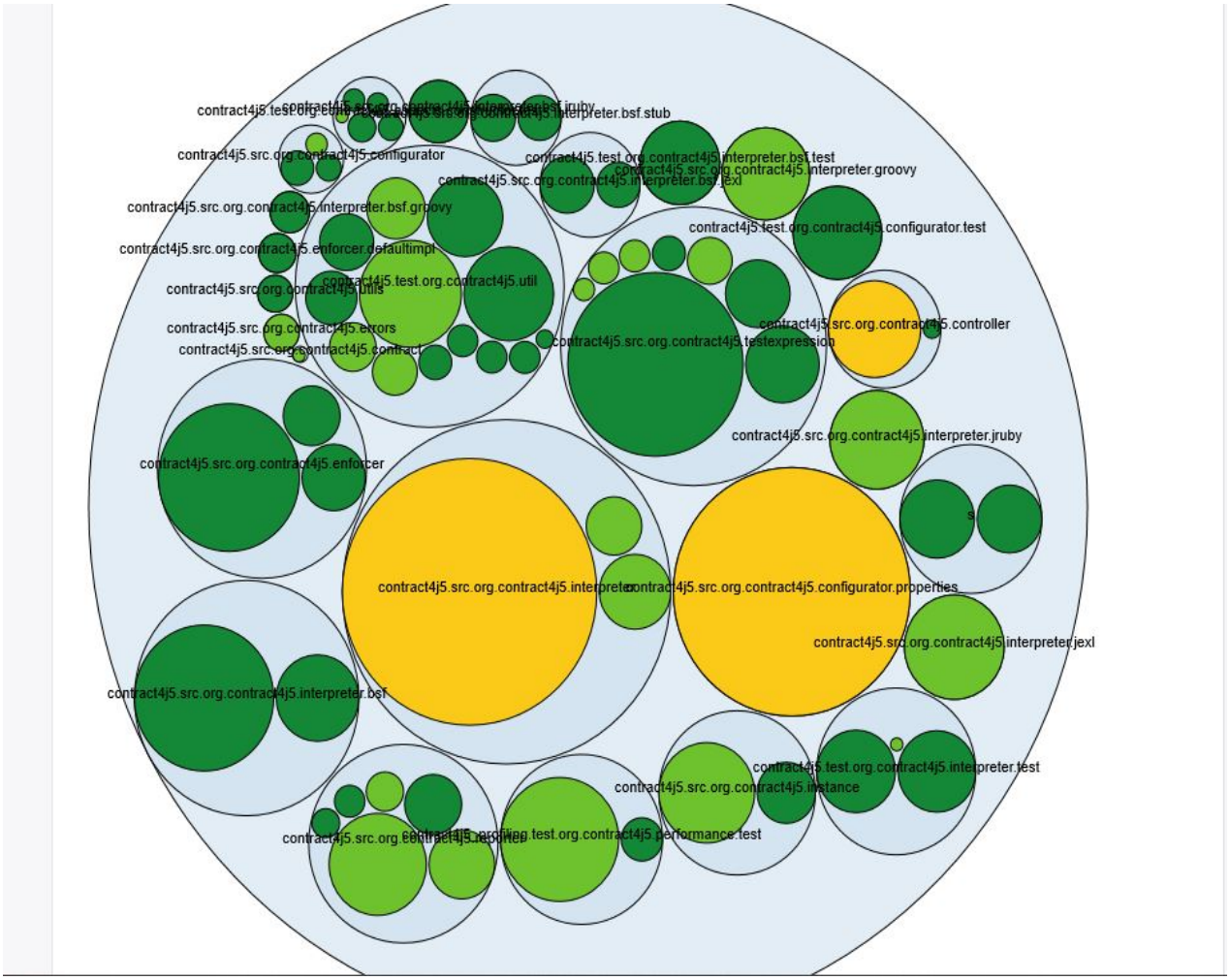


Figure 4.18 Metric distribution packages Contract4J5 project

The Contract4J5 project has a medium-high value for size 29% of the total source code, complexity 29%, lack of cohesion 2% and WMC is 29%. It has small coupling values, which means well structured and has good design, as shown in Figure 4.19.

.

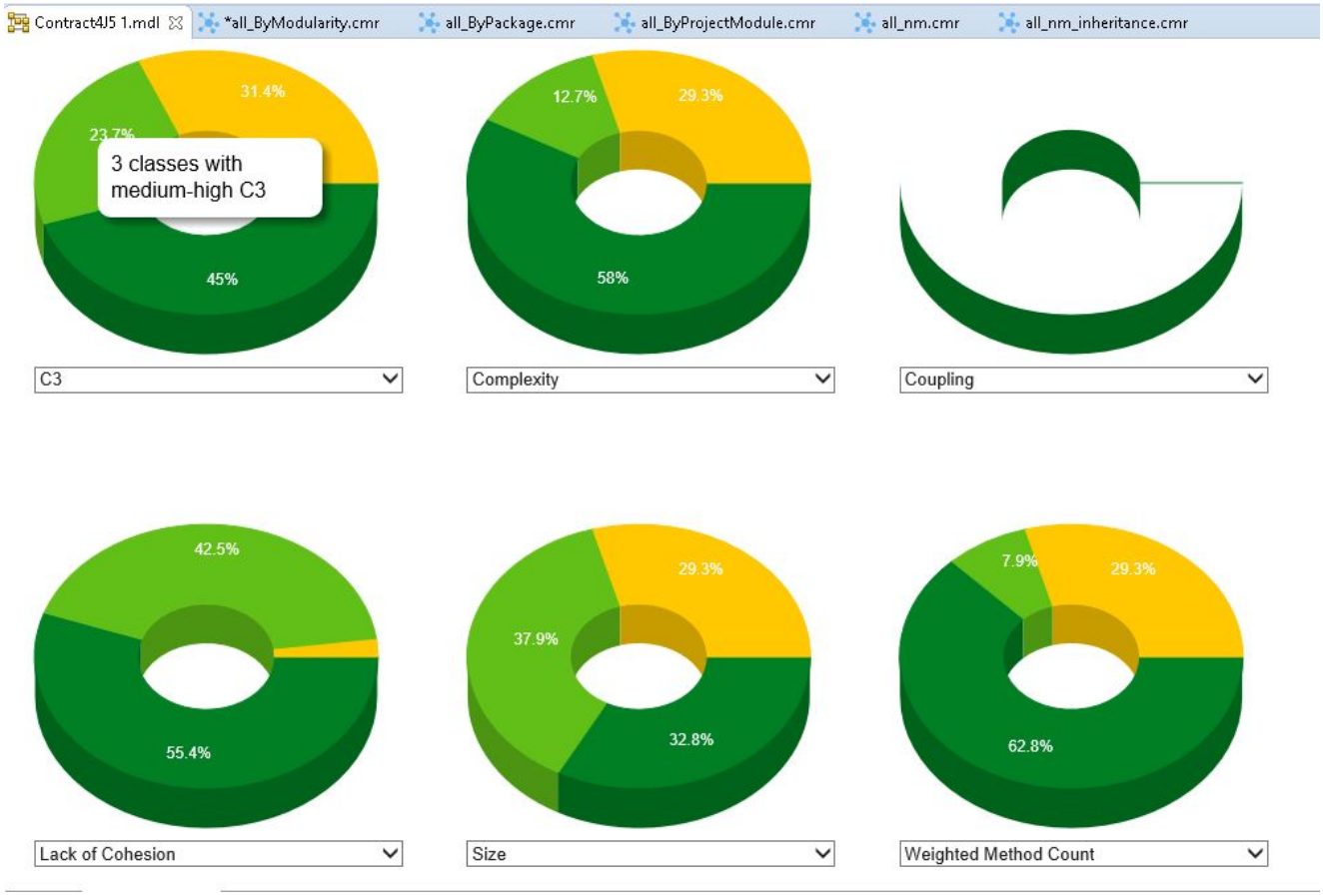


Figure 4.19 Six Average values Metrics for Contract4J5 Project

The interaction between the classes in the project has been representing as it shows in Figure 4.20. Circular shape and size represent the package size, and the colour represents metric values. Arrows show classes, communications and the colour of the arrows represents the metric values. The metrics values show that there is an average interaction between classes in the project, which represent a normal quality attributes between the project components.

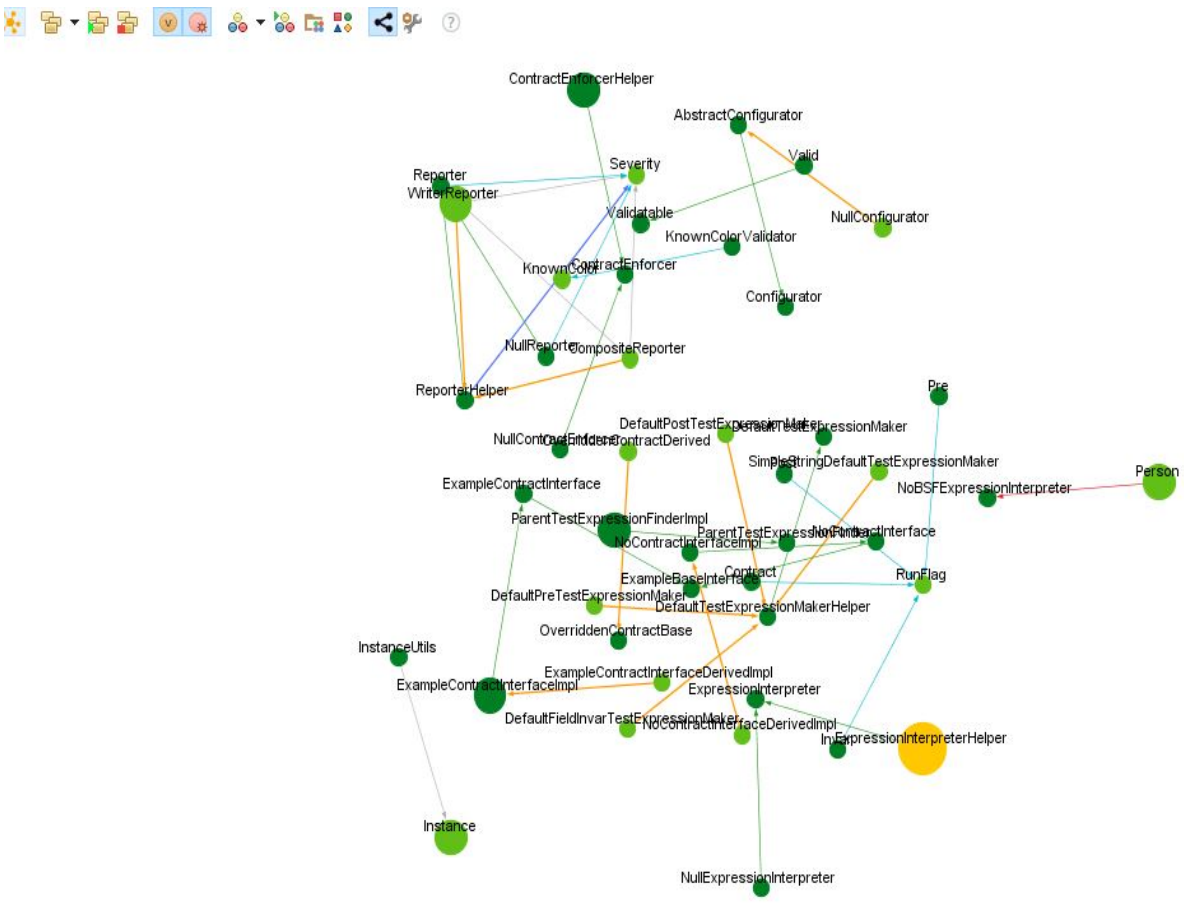


Figure 4.20 Classes relationships in the Contracy4J5 project

The Metric tool is so powerful it has much other functionality. It shows many metrics reports as the web interface. The General information about distribution quality attributes is represented in Figure 4.21 with the excellent virtualisation technique.

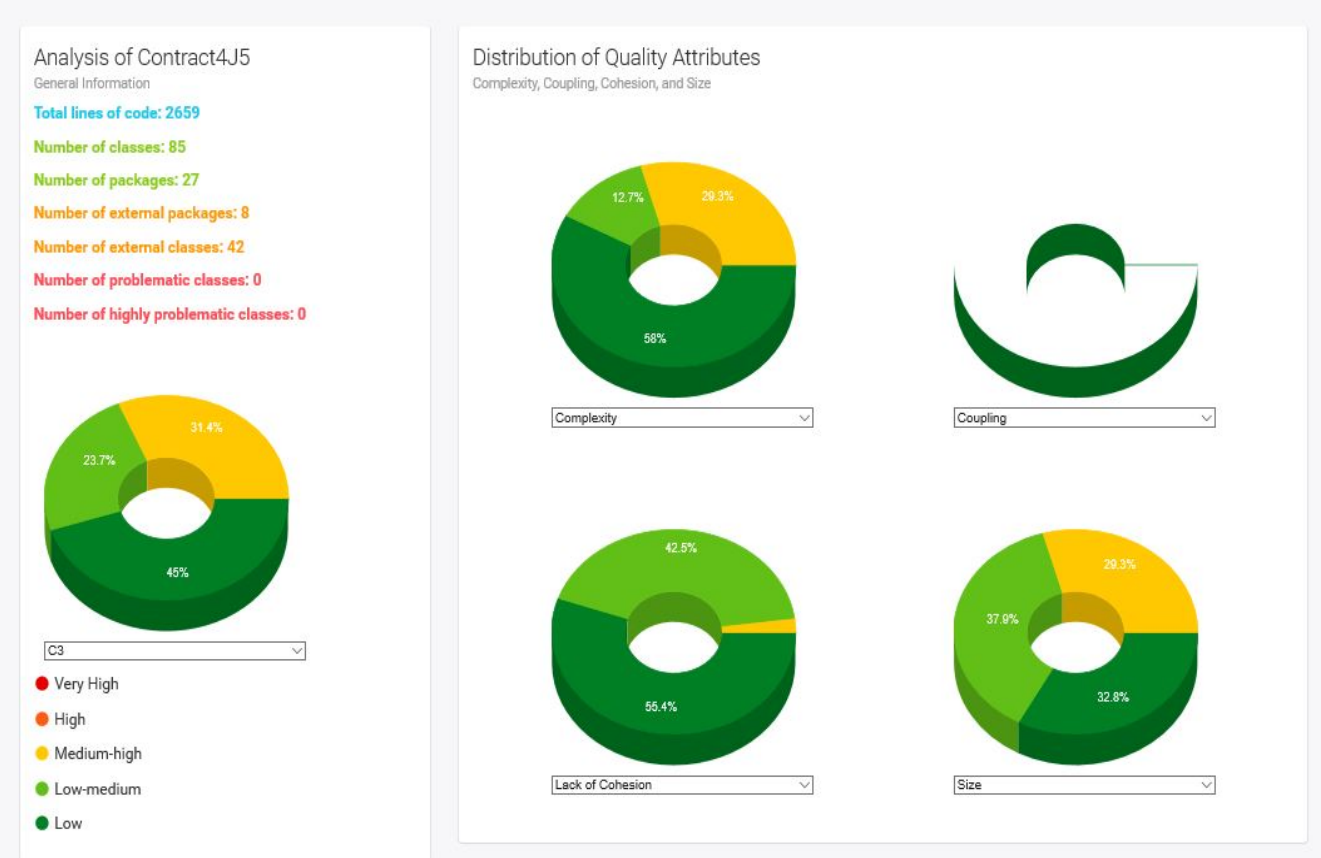


Figure 4.21 distributions of Quality Attributes for Contract4J5 project

This project has 143 java files and 14 aspects files with extension Java and aj consequently. It is a mid-sized project. The part of the appearance code line has nearly 63% of the total code line. However, the CDC and CDA have a significant impact on modularity parameters and other parameters, as it shows in Table 4.17, 4.18. The portion of the Line of Code of Aspect has almost %63 percentage of the total Line of Code. However, CDC and CDA have a significant impact on modularity metrics and other metrics, as explained in table 4.17, 4.18.

The metrics has been categorized the extracted metrics on subcategories as it appears in the table 4.17 below:

Main components of metrics	Sub- components of metrics
Complexity	Weighted Method Count (WMC)
	Depth of Inheritance Tree (DIT)
	Response For a Class (RFC)
	Specialization Index (SI)
Coupling	Number of Children (NOC)
	Coupling Between Object Classes (CBC)
	Access to Foreign Data (ATFD)
Cohesion	Lack of Cohesion of Methods(LCOM)
	Lack of Cohesion Among Methods (LCAM)
	Lack Of Tight Class Cohesion (LTCC)
Size	Line Of Code (LOC)
	Number of Fields (NOF)
	Number of Static Methods (NOSM)
	Number of Methods (NOM)
	Number of Static Fields (NOSF)
	Number of Overridden Methods (NORM)

Table 4.17 OOP Metrics and some AOP Metrics categorization Contract4J5 project

All the metrics values need to be normalizing to unifying the outcomes results using the following equation:

$$\boldsymbol{EQ\ 4.18 \ldots . Nor(x) = \frac{(x - \min(x))}{\max(x) - \min(x)}}$$

Main Components of Metrics	Sub- Components of metrics	Total values	Normalized values Nor(x)
Complexity	WMC	11.441	0.328
	DIT	0.986	
	RFC	6.613	
	SI	0	
Coupling	NOC	0.32	0
	CBC	0.2	
	ATFD	0.013	
Cohesion	LCOM	0.121	0.006
	LCAM	0.249	
	LTCC	0.175	
Size	LOC	52.137	1
	NOF	0.8	
	NOSM	0.4	
	NOM	3.986667	
	NOSF	0.293	
	NORM	0	

Table 4.18 OOP Metrics and some AOP Metrics using Code-MR plug-in for Contract4J5 project

4.7.2 AOP quality metrics extraction using java written program:

- **Crosscutting Degree of an Aspect (CDA) metrics**:

  The equations of CDA metrics as following

  CDA = Total of Effected Methods / Total of the Pointcuts

  In Contract4J5 project, number of aspects are 34, number of Pointcuts are 26 that have called 105 methods, so the CDA is

  EQ 4.19..... CDA = Total of Effected Methods / Total of Pointcuts

  =105 / 26

  =4.03

  Another indication of finding the effect of CDA on method is by using the following formula (CDA/WOM)

  CDA/WOM =4.03/11.45, which is equal to 0.35 and indicates the Advice effect on several modules.Contract4J5 shows high values of the CDA, which means many modules are affected by Aspects of Pointcuts and intertype declaration.

- **Coupling on Advice Execution (CAE) metrics**: In the Contract4J5 project, CAE Metrics measure by counting the number of Advice. The total Advice is 10; therefore, the CAE value is 10. Another indication is by using the following formula CDA/WMC.

  EQ 4.20..... CDA/WMC =10/11.45

= 0.87

This result indicates the Advice execution effect on several methods in the project. Contract4J5 shows the average values of the CAE, which means many modules, are affected by Advice excursions.

- **Concern Diffusion over Components (CDC) metrics**: The project Contract4J5, we can calculate by using the following formula

  EQ 4.21..... CDC = number of components that assist in implementing a concern + number of components that access the components. The value of the CDC of this project is 128, which means it has high CDC values among the project; there are 143 java classes and 14 aspect files. If we take an average effect overall the project, this will be CDC Effect = CDC values /Total number of Classes and Aspect =128/157 =0.81

- **Lack of Concern-based Cohesion (LCC) metrics:**

  LCC metrics' value is calculated by counting number of concerns. The project Contract4J5 contains ten concerns implementing methods from the abstract classes, six concerns that are implementing methods, extending abstract classes, and implementing Pointcuts. This project is cohesive and has high values of the LCC metrics project regarding the number of aspect project sizes.

- **Aspect complexity (AC) metrics:**

  There are two types of Aspect Complexity metrics, Low-level and High level. Low-level Metric has measured the signature and calling methods. High-level AC values are measured: Pointcuts, Aspect, intertype declaration, function calls, before, after the keyword, and || and && operators. The results AC are 140, which show that the Aspect of this project has high values of Aspect complexity. Table 4.19 below provides details on the AOP measures for the Contract4J5 project. Normalisation values range is between 0 and 1. Size, quality parameter is has maximum number which is equal to 1. Mean values AO parameters for complexity are 0.47, showing the mean value, which is considered a non-complex project. The project has low coupling values 0, which means the right structure for this project. The low cohesion values of this project have shown that it lacks high readability and maintainability. Modularity values are 0.38, showing that the system has a less impact on the connection between other system modules. It shows the number of modules that are independent of another module of the system.

	Size					Complexity					Coupling			Cohesion		Modularity	
No.	Project Name	Number of Class files	Number of Aspects files	Line of code aspect	Number Pointcuts	No. Of Aspects	WMC: Weighted methods per class	DIT: Depth of Inheritance Tree	RFC: Response for a Class	Cyclomatic Complexity of Aspect CCA (Advice, Pointcuts, JoinPoint)	CBO: Coupling between object classes	NOC: Number of Children	Coupling on Advice Execution (CAE)	LCOM: Lack of cohesion in methods	Lack of Concern-based Cohesion (LCC)	Concern Diffusion over Components (CDC)	Crosscutting Degree of an Aspect (CDA)
5	Contract4J5	143	14	112.214	26	34	11.45	0.98	6.61	140	0.2	0.32	10	0.12	16	128	4.03
Total Values for each factor	329.214					159.04					10.52			16.12		132.03	
Nor(x)	1					0.47					0			0		0.38	

Table 4.20 Bash Script & java program AOP metrics Extraction for Contract4J5 project

4.7.3 Discussion

The metrics values of (AO/OO) have extracted using two techniques: Code-MR tools and developed java program. Therefore, Tables 4.18 and 4.19 present some findings. The combined tests gave approximately the same result. Table 4.20 below have been showed the average values of AO/OO metrics.

	Size	Complexity	Coupling	Cohesion	Modularity
Average Values	1	0.39	0.0	0.003	0.38

Table 4.31 OOP/AOP metrics values for Contract4J project

Table 4.19 are showing the average AO/OO metrics values between the various parameters were clarified. In all measurements, both tables have similar outcomes. The metric size value is equal to 1, which is the maximum value of the size measures. These findings show Contract4J5 is not a complex project. The total complex parameter values are 0.39. These findings showed that the OOP and AOP source code are less complex in this project. With less complexity, maintenance and testing costs are reduced. The coupling values are very low, almost zero, and the cohesion of the AO/OO measurements is greater. These findings suggest that low coupling involves additional separation of independent source code. Higher cohesion values mean that a portion of the source code at a moment in the code base is connected. Modularity values are 0.38, regardless of the low modularity between the modules of the system. Modular metric values show how many modules in the system are independent of another.

4.8 Conclusion

This has been explaining the method of extracting AO/OO metrics and the reflection on AO/OO quality metrics. Five case studies from AO/OO technologies have been used to get the proposed measures. Relations between object-oriented and aspect-oriented measurements have been discussed. Chapter explains the common measures between OOP and AOP in the hybrid application development techniques. The software quality framework for OOP has been described using several examples. The weakness and strength of the OOP measures is described.

The chapter showed that AOP quality measures feed into the proposed new framework of hybrid AO/OO application systems. Project size will vary depending on different development techniques, project requirements, and user needs. Metrics outcomes have been difference, normalisation techniques have been used to unify the results. Code-MR and written java tools have been used as a combined extraction measurement method for this project. The ultimate results showed that the AO and OO measures are related. In conclusion, the results showed that AOP quality measures will affect the size, complexity, modularity, coupling and cohesion in hybrid application projects. Effects of quality measures are positive for coupling cohesion and modularity and negative for size and complexity. Data extraction from theses five projects has been categorised in software quality metrics of hybrid application systems. The proposed model has been designed in chapter 5, section 5.6. The evaluation chapter 6 will explain these findings.

CHAPTER 5

SOFTWARE QUALITY MODEL

5.1 Introduction

This chapter describes how to integrate the AO quality metrics into the AO/OO hybrid quality framework. It concentrates on the research on static metrics and a few dynamic metrics to measure the development of hybrid AO/OO systems. The chapter also discusses basic OOP measurements [78] in hybrid AO/OO development measures and discusses the typical relationship between OOP and AOP measurements. Quality is an essential property of software systems and normally refers to the degree to which a software system meets the expectation that its specifications will be met. Quality values are defined by attributes such as modifiability, durability, interoperability, portability, health, predictability, extensibility. IEEE STD 1061 (Standard for Software Quality Metrics method) describes quality. The IEEE STD distinguishes between quality and quality attributes (such as performance, ease of use, and maintenance). Software quality is the extent to which the software has a desirable combination of attributes [26]. A software quality template describes a set of properties related to software quality and its relationships. The relationship can be quantitative or qualitative. Software quality model documentation and is often combined with software quality assessment metrics [27]. In order to assess the qualities of an application, it has needed at least a quality model, which will evaluate all the application's features. There are several quality management models have been developed to test general and specific categories of software products. These models were proposed for determining the general or limited scope of software products. Software measurement is an activity that takes place at all stages in the software development process. Many intermediate or end software products are produced and

evaluated using the parameters of the software products in this process. One of them is the source code of the program, which is part of the final software system and is evaluated for quality. Metrics for Aspect Oriented applications have been showed the advantages and disadvantages of cross concerns concepts. New Product quality model for hybrid application software has been proposed

5.2 Justification and Significance

Determining the quality of hybrid AO/OO applications based on the corresponding source code attributes is a critical quality assurance and control function. It is essential to have a standard technique of evaluation methods that can aid the measurement evaluation procedure. Several tools extracted static measurements for AOP and OOP systems separately without considering the relationships between those measurements. The preliminary analysis and review of the state of the art in the context of software quality does not reveal a wider recognition of the standard quality framework for the hybrid AO/OO application [141]. Techniques and methods for extracting product metrics from the AO/OO hybrid application system are limited and inappropriate. Sometimes the different tools provide conflicting results with the same metric formula and help only some relevant metric choices. This is because the estimate of the measure may be viewed differently with different methodological techniques [142], Therefore it is essential to develop a framework for extracting AO/OO product metrics based on ISO 9126 quality framework [152][153][154][156][157][158].

5.3 A framework for Aspect Oriented Software Quality Model

There are many architectural models available for developing a software system. Modules and objects oriented OO approaches are the most commonly used for developing a software system. However, cannot consider all requirements of current and complex software systems [127]. According to Alexander [128], he stated the advantages reduce the amount of written code and greater cohesion reduces the complexity of the system. Internal characteristics can measure quality of the AO system such cohesion, coupling and complexity [129]. The metric is a qualitative indicator of any attribute of the software system and the model specifies the relation between these metric. Aspect Oriented approach is a new paradigm and has a different abstract and is not autonomous [130]. The Aspect modules are built in with AO modules or with OO classes. Therefore, most of the software quality components/sub-components of ISO/IEC 9126 can be measured with AO/OO quality framework with additions/modifications. Because of the new type of abstraction in AO technology, some new software quality features/sub-features should be added, describing the new features of AO technology. So we have to come up with an ISO/IEC 9126

quality model. Modularity, code reduction, complexity and re-usability were added as new sub-quality features. The sub-components have a direct effect on the components such as efficiency, maintainability, functionality, usability respectively.

5.4 AOSQUAMO Aspect Oriented Quality Model

The Aspect Oriented Software Quality Model (AOSQUAMO) is a derivative from ISO/IEC 9126 quality model. Complexity, modularity, reusability and code reduction have been proposed as new quality sub-components under major components such as usability, maintenance of performance and functionality,[143] [144]. A quality assessment of the AO system applied to the existing model using the Analytical Hierarchy Technique (AHP) [145]. The AHP has been widely adopted in cross criteria decision making and has been successfully applied to many practical decision-making problems [143]. The fundamental properties of quality attributes of the AOSQUAMO are listed below [143] [144]:

a) Modularity: AOP deals specifically with differentiation of concerns in the design of applications. It supports modular programming, in particular by integrating issues that overlap the modular nature of conventional programming methods. For example, login policy implementation it shows how the code is distributed across a collection of classes and methods to enforce the policy. The code applying the login policy is extracted from all the AO method classes and is implemented in a module called Aspect. Aspect therefore locates the code which affects multiple classes and methods in a unified location [132]. The modularity of overlap concerns would be beneficial where improvements or adjustments are necessary in the code related to overlap concerns. With these changes, the impact would be reduced. That means modularity boxes support system maintenance, therefore it has been proposed the integrate modularity as a sub-component below the maintainability characteristic.
b) Complexity: The complexity of Aspect Oriented software systems is divided into the complexity of the code, which can be described as static metrics, and the complexity of the interaction, which can be described as dynamic metrics.Code complexity is related to complexity attributes, the complexity of the operations (methods, Advice, Intertype declaration and the complexity of the embedded classes/aspects. The interactions between the class and the aspect have been done because of implicit/explicit invocations of operations, the appearance of exceptions, comparisons of attributes, etc. Kumar et al. [143]

have identified some complexity measures for AO systems. The complexity function could also help estimate the effort required to build and maintain the software system.

c) Reuse: There are three levels of reuse: functional, code property, and reuse of the weaving process. Functional re-usability of the Object Oriented approach is like the Aspect Oriented approach. The reuse of codes in the OO approach is because of their heritage rights. The code related to Cross Cutting concerns are modularised into one Aspect of the AO approach, which was scattered into primary concerns classes. This code will later be integrated with the classes which have to support this policy through a process known as weaving. Weaving has injected the aspect code into the syntactic structure of a primary concern in well-defined places (Join points) and called reusability of element. Although in this study, the proposed AO quality model proposes to add re-usability under the characteristic function.

d) .

e) Code Reducibility: Code reducibility means the use of AO techniques to reduce the written OO code, for example, spread across several classes which are implemented in AO technology in a module. This causes the amount of written code is decreased. All codes that would otherwise be distributed in a primary collection are now localised, reducing duplication, Reducing the amount of code would without a doubt increase system performance in terms of time and memory. Some suggest that adding code reducibility under characteristic of efficiency [144].

f) Most of the quality models offered have the fundamental characteristics of quality, functionality, reliability, usability, efficiency, maintainability, and transferability. When selecting the sub-characteristics of these characteristics, researchers have different opinions on the sub-characteristics that feed the key characteristics of the quality model components. The details of the common subset are.

g) Feature: The fundamental purpose of any product is a feature. It is a set of attributes that focus on a set of functions and the properties defined. When used under specific conditions, classes and aspects shall provide functions and services according to the requirements. The current class features can be easily changed/improved by inheritance, and the Aspect-oriented approach can easily change the crosscutting features. This will reduce costs and speed up product development.

h) Suitability: Relevance explains the degree to which the item mixes into classes. Aspect involves class integration; developers should know increasing the complexity if adding Aspect to the program.

i) Accuracy: It tests whether the program provides reliable results with the correct accuracy after implementing Aspects when used in various conditions.
j) Interoperability: The capacity of various systems and organizations to work collaboratively. The interoperability of the AO and OO structures amounts to a combination of class aspects.
k) Compliance: These subcomponents explain whether an AO system meets an international standard or certification.
l) Security: The security aim requires the protection of information and data from fraud, misuse, or prevention of unauthorised access to the network. Security resources may be considered a cross cutting issue because security policy must be applied when it is appropriate. This security concern related code could be implemented in one area, which helps to enhance the security of the source code.
m) Reliability: The reliability characteristic determines its ability to support the provision of its services over a specified period under defined conditions. If it is not run on time, its probability AO system will cause a failure in a period.
n) Maturity: These sub components explain the amount of work performed with current development tool technology. Compared to OO technology, AO technology requires more to attain high maturity.
o) Failure Tolerance: Describes whether the AO system can maintain a specified level of performance in the event of failure.

This is an important feature to understand within the context of the AO system. The failure of the AO system can take place in. The defect is an emergent property resulting from some interaction of the Aspect with the primary preoccupation of four different stages:

The defect lies in a part of the major preoccupation, which does not affect the woven component.
The fault lies in the aspect-specific code, which is isolated from the woven background.
Recoverability: This shows the capability of the AO software to restore its output level and recover the specifically affected data when a malfunction occurs.

1. Usability: it means how easy to use of a class or feature, for instance, an AO program may be used and run during operation under various conditions. The developer can only use aspect when the look and classroom knowledge is combined.
2. Understand ability: This feature explains the software's ability to allow users to consider the adequacy of the software and its use for particular tasks and conditions of use. In the AO sense, the ability of developers to solve the underlying primary concerns, apply the appearance code and integrate the appearance code with the primary concerns.

3. Learning ability: This sub component describes the ability of the AOP program that can help users understand and use applications.
4. Operability: This sub component defines the AO software's ability to allow users to operate and manage it.
5. Efficiency: it concerns with how the system resources have used to provide the required functionality are related to this component. For instance, disk space usage, network, processor time, etc.
6. Maintainability: This feature describes how to change the class or aspect of the system or any functionality. AO technology has an advantage over OO technology for a system with transversal issues in the maintenance of transversal issues. Modularity of cross-cutting concerns was introduced as a subtype of modularity. It is characteristic of maintainability, which states that the maintenance of such a system would require fewer efforts than OO.

.

Other futures have been described as:

I. Analysis: This sub-feature defines the capability of the AO software to be evaluated for defects or causes of defects in the AO software. It can detect changes in primary concern, aspect or process visualisation, i.e., integration of aspects with primary concerns.
II. Editability: This sub-feature describes AO software's ability to implement a specified modification. Grover et al. [137] [138] provided guidance for the measurement of variability in AO software systems.
III. Stability: This sub-feature refers to the ability of the AO program to mitigate the unintended effects of software changes.
IV. Testability: these sub-components concerns about ability of the application to validate changed software.
V. Portability: This function refers to how the program can manage changes in its surroundings or specifications. The extent to which this feature exists in a particular system may contribute to Object Oriented design and implementation practices. Aspect Oriented architecture and application in terms of portability are the same as Object Oriented, as the world would be the same for appearance and object.
VI. Adaptability: This is the functionality of the AO software which may apply to various specified platforms. Despite the platform's independence, the expanded AO functionality from Java provides greater adaptability.
VII. Installer: The AO software can be installed easily on different platforms.

VIII. Replaceability: This is the capacity of AO application that can run on difference environments.

IX. Complexity: it concerns with WOM (Weighted Operations in Module), the number of operations within a module influences internal complexity. DIT (Depth of Inheritance Tree): is measure the longest path between a particular module and the root of the Aspect/Class hierarchy. The higher an Aspect/Class is in the hierarchy, the greater the number of operations it inherits and making it more complex to understand and change measured.

X. The difference between the quality specifications of ISO/IEC 9126 and the Aspect Oriented Software Quality Model (AOSQUAMO) has been explained in Figure 5.1. Four salient features have been proposed, and these have many sub-characteristics and measurements that contribute to these key features.

Quality Type	Characteristics	Sub-characteristics
Software Product Quality	Functionality	Suitability
		Accuracy
		Interoperability
		Compliance
		Security
	Reliability	Maturity
		Fault tolerance
		Recoverability
	Usability	Understandability
		Learn-ability
		Operability
	Efficiency	Time behavior
		Resource behavior
	Maintainability	Analyzability
		Changeability
		Stability
		Testability
	Portability	Adaptability
		Install-ability
		Replace-ability
		Conformance

Quality Type	Characteristics	Sub-characteristics
Software Product Quality	Functionality:C1	Suitability:SC11
		Accuracy:SC12
		Interoperability:SC13
		Compliance:SC14
		Security:SC15
		Reusability:SC16
	Reliability:C2	Maturity:SC21
		Fault tolerance:SC22
		Recoverability:SC23
	Usability:C3	Understandability:SC31
		Learn-ability:SC32
		Operability:SC33
		Complexity:SC34
	Efficiency:C4	Time behavior:SC41
		Resource behavior:SC42
		Code-reducibility:SC43
	Maintainability:C5	Analyzability:SC51
		Changeability:SC52
		Stability:SC53
		Testability:SC54
		Modularity:SC55
	Portability:C6	Adaptability:SC61
		Install-ability:SC62
		Replace-ability:SC63
		Conformance:SC64

Figure 5.1 a. Quality Characteristics of the ISO/IEC 9126, 2004. b. Aspect-Oriented Software Quality Model

COTS are a quality model that has added new components to its quality model as stakeholders responsible for developing, maintaining, and integrating. This model emphasises measuring the quality of the system based on the components. [139]. in this model, the sub-components have divided into duration and life-cycle categories depending on their nature. They also added capacity as a subtype to the feature, showing if the older version of the component is compatible with its current version.

5.5 Proposing Aspect Oriented Product Quality Metrics:

Proposing new product quality models should cover all the functionality of the AO software system; it needs to examine the new functionality and limitations of the new technology. AO programming languages are also added into a native programming language. Several available technologies, such as Aspect C, extend C. Aspect C++ extends C++, AspectJ extends Java, CaesarJ extends Java, and Aspect XML extends XML,. Since AO technology cannot exist on its own, it will have all the technology features from which it is derived. Given that AO technology cannot exist on its own, it will have all the technological characteristics from which it arises. For example, AspectJ contains all the functionalities of Java and other functionalities added to AspectJ. If the AO technology is derived from the OO technology, then it will have all the characteristics of the OO technology and additional functionality added to the aspectual code. Different quality components/sub-components should be added, which may cover new features of cross cutting concerns of Aspects and their integration with primary concerns the classes. There is also a need to redefine existing features/sub-features within the context of AOP technology. As it is described in table 5.1.

Quality Type	Characteristics	Sub-Characteristics
Software Product Quality	Functionality	Suitability
		Accurateness
		Interoperability
		Compliance
		Security
		Extensibility
	Reliability	Maturity
		Fault tolerance
		Recoverability
		Complexity
	Usability	Understand ability
		Learnability
		Operability
	Efficiency	Time behaviour
		Resource behaviour

.

		Service loading behaviour
	Maintainability	Analyzability
		Changeability
		Stability
		Testability
		Adaptability
		Code Weaving
	Portability	Installability
		Conformance
		Replaceability
		Co-existence
		Re-usability

Table 5.1 Proposing AOP software Quality Model

5.6 New Product Quality Framework for Hybrid application system

Various components of the ISO 9126 quality framework have been proposed. It shows in Figure 5.2, below the blue rectangle, the new sub-characteristics proposed for measuring the quality of the hybrid application system, for instance (extensibility, complexity, service loading behaviour, code weaving, and Re-usability). These software product quality sub-characteristics have sub-characteristics that show with proposed new quality metrics.

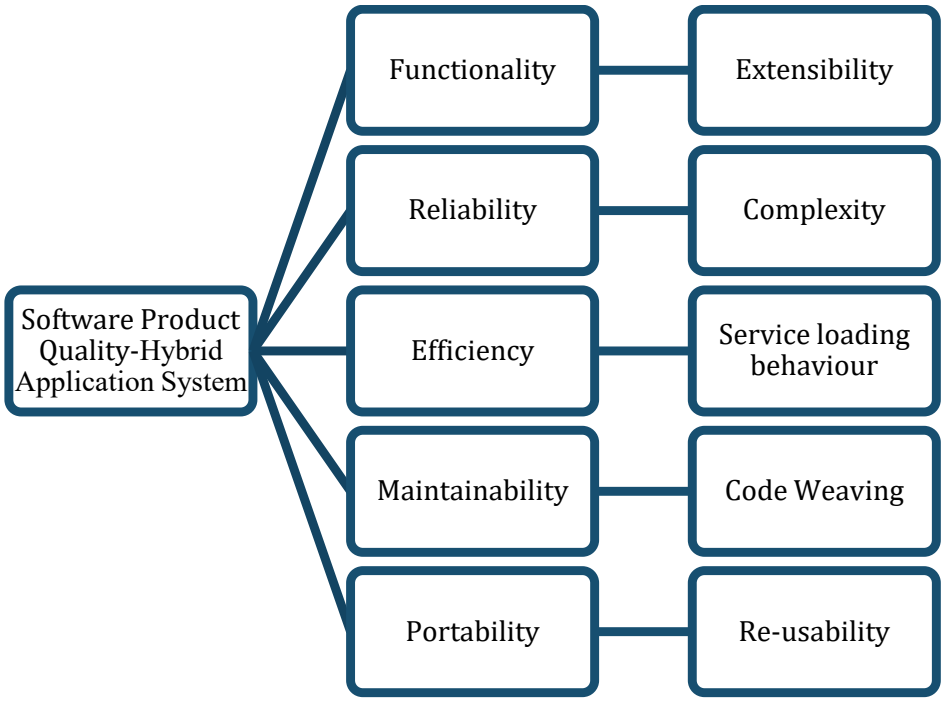


Figure 5.2 Software Product sub-characteristics

5.7 New Quality Metrics for Hybrid application system

New quality metrics for the Hybrid Application System attributes, These main components and sub-components of the proposed quality measure model for hybrid application systems [145]. It is proposed to build on the AOP data quality measures in Chapter 3 of the extraction, Section 3.7, as shown in Figure 5.3 below:

- Extensibility: can be measured by the following metrics:
- Crosscutting Degree of an Aspect (CDA)
- Joinpoints Extension (JPE)
- Concern Diffusion over Components (CDC)
- Number of Models
- Complexity can be measured by the following metrics:
- Weighted Method Count (WMC)
- Depth of Inheritance Tree (DIT)
- RFC Response for a Class (RFC)
- Responses for Aspect (RFA)
- Number of Aspects (NOA)
- Service Loading Behaviour (SLB) can be measured by the following metrics:
- Line Of Code (LOC)
- Number of Methods (NOM)
- Number Of Fields(NOF)
- Weighted Joint points Counts (WJC)
- Line Of Code Aspect (LOCA)
- Code Weaving (CW) can be measured by the following metrics:
- Number of Children (NOC),
- Coupling Between Object Classes (CBO)
- Access to Foreign Data (ATFD)
- Coupling On Advice Execution (CAE)
- Re-usability can be measured by the following metrics:
- Lack of Cohesion Of Methods (LCOM)
- Lack of Cohesion Among Methods (LCAM)
- Lack of Tight Class Cohesion (LTCC)
- Lack of Concern Based Cohesion (LCBC)
- Lack of Cohesion in Joinpoints (LCJ)

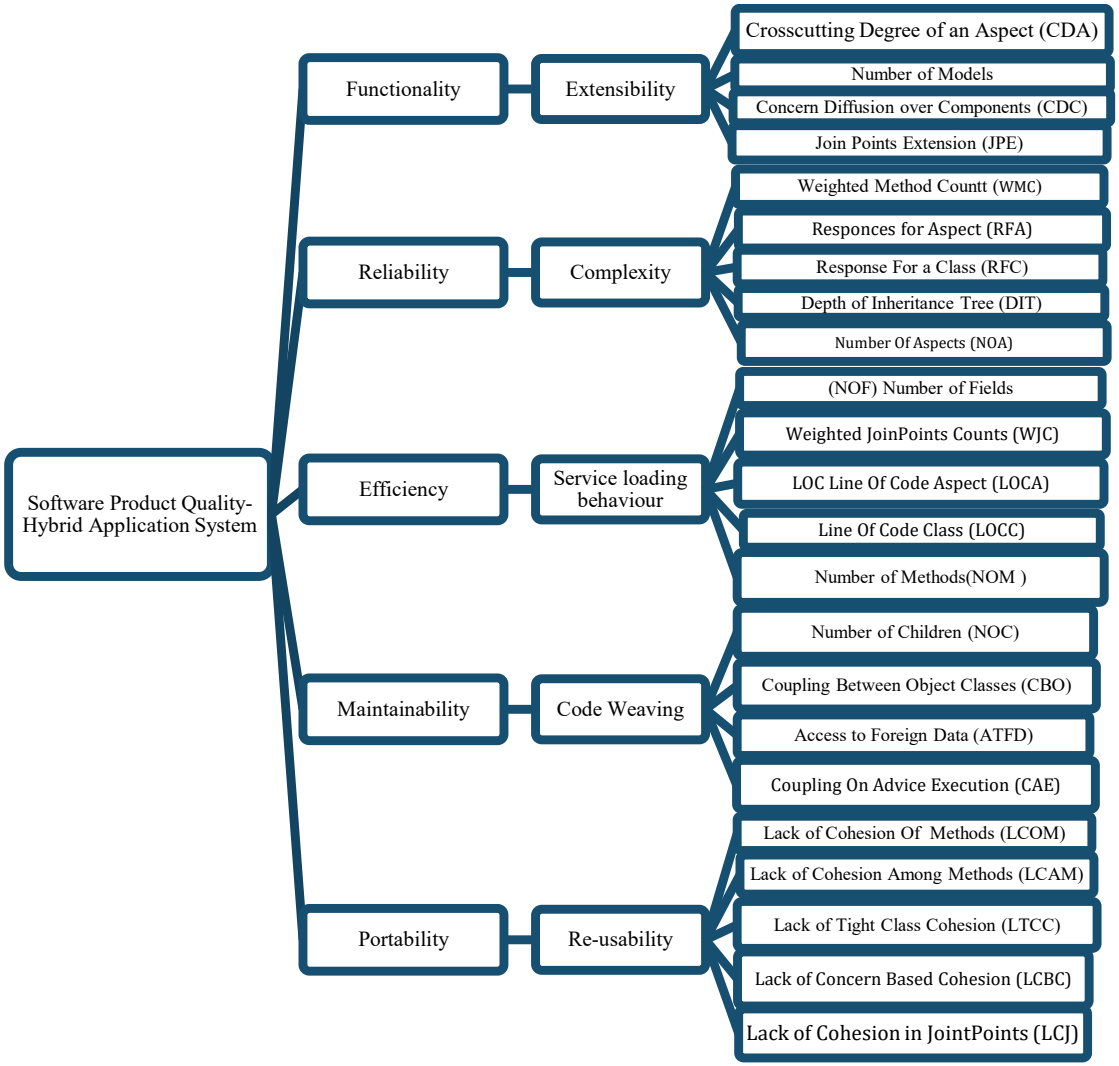


Figure 10 Sub-characteristics for proposed software quality framework for Hybrid AO/OO Systems

5.8 Conclusion

This chapter explained the various software quality models. It describes the history of various types of software quality models, such as McCall, Boehm, ISO/IEC 9126, and another model which relates to software quality measurements. It also highlighted the importance of offering a new framework software quality model for measuring AO/OO software. This chapter also describes several works that were done to measure the quality of the AOP software. Proposed framework is based on ISO/IEC 9126 with some sub-specifications for the software quality model. Sub-characteristics represent the proposed actions discussed in Chapter 3. These parameters and its sub-characteristics of the quality model will be evaluated in Chapter 6. The proposed aspect-based product quality framework will focus on the AOP part and the new AOP measures. These new measurements were focused on static measurement, and few dynamic measurement of product quality, as dynamic measurement requires a team of developers, product managers and users. This study comprises five different projects developed using the AO/OO code. The primary aim was to show the impact of AOP metrics on the quality of software products using the proposed framework

.

CHAPTER 6

PRODUCT QUALITY METRICS FOR HYBRID APPLICATION SYSTEM AO/OO

6.1 Introduction:

This chapter summarises the assessment process. The metrics of AO and OOP have been defined and standardised for the proposal of new hybrid application metrics. The combination of both AOP and OOP metrics has been reflected in Hybrid Application Quality Model. This data was generated using plug-in Code-MR tools to extract common statistical measures from AOP and OOP and writing a Java program to extract AOP measures only. The evaluation of the combination of these metrics was conducted using the IBM AMOS tool [145][147][148]. AMOS has a powerful program that can accept SPSS or Microsoft Excel data files as input. AMOS can perform many statistical equations to produce different results in terms of their relationship between the quality model variables. Quantitative data collection has been conducted, as explained in Chapter3 which are need to evaluate them. This chapter has used statistical analysis to validate the extracted data and prove the proposed hypothesis. The model was designed using structural equation modelling, as discussed in Chapter 5. Five hypotheses were considered, each hypothesis contains many measures which influence this hypothesis as it is described below.

.

I. Hypothesis 1 (Extensibility) the measures of H1 had an impact on Extensibility (CDA, JPE, CDC and NOM). Extensibility will affect the functionality of the Quality model.

II. Hypothesis 2 (Complexity) H2: This will have an impact on the reliability of the proposed hybrid quality model. The Complexity in the model have measured by five metrics (Weighted Method Count (WMC), Depth of Inheritance Tree (DIT), RFC Response For a Class (RFC), Responses for Aspect (RFA), Number Of Aspects (NOA)) Complexity has been a negative impact on reliability. As complexity increased, system reliability was reduced.

III. Hypothesis 3 (Service Loading Behaviours) Service Loading Behaviours has been measured by these metrics (Line of Code (LOC), Number of Methods (NOM), NOF (Number of Fields), Weighted Joint points Counts (WJC), Line of Code Aspect (LOCA)).

IV. Hypothesis 4 (Code Weaving) H4 Code Weaving has been measured by these metrics (Number of Children (NOC), Coupling between Object Classes (CBO), Access to Foreign Data (ATFD), and Coupling on Advice Execution (CAE)).

V. Hypothesis 5 (Re-usability) H5 Re-usability has been measured by these metrics Lack of Cohesion Of Methods (LCOM), Lack of Cohesion among Methods (LCAM), Lack of Tight Class Cohesion (LTCC), Lack of Concern Based Cohesion (LCBC), Lack of Cohesion in Joint Points (LCJ). The whole evaluation process makes it clear that the aspect-oriented software quality model is acceptable or not.

6.2 Evaluation using Structure Equation Modeling:

Structural Equation Modelling (SEM) is a multivariate procedure is used many analytical methods to analysis structure relationships. The evaluations have been conducted using different techniques to provide quantitative results of a hypothetical model estimated by the analyst. Several related latent variables can be tested in SEM using statistical techniques. The purpose of the SEM is to determine the extent to which the sample data support the postulated model. If the sample data supports the hypothetical model, then more complex hypothetical models may be theorised. If the sample data does not confirm the hypothetical model, either the first model can be changed and tested, or other hypothetical models need to be created. Therefore, SEM tests hypothetical models using analytical technology for theoretical tests and understanding of complex relationships between variables. SEM can test various kinds of hypothetical models [160]. SEM has capabilities of design and defines many variables in the evaluated model. There are two important kinds of variables: latent's and observed variables. Latent variables (constructs or factors) cannot legitimately be detected or estimated [161]. Latent's are observed or estimated indirectly. They are collected from many observed variables that have been measured through tests, examinations, etc. An independent variable is a variable that isn't affected by some other variable in the model and not assign to it [162]. Whether observed or latent, variables can also be characterised as independent or dependent variables.

6.2.1 Structure Equation modeling common terms:

SEM extends the overall linear model which enables a specialist to simultaneously test a group of regression equations. The reason for SEM is to analyse a lot of connections between at least one exogenous variable (independent variable) and at least one endogenous factor (dependent variable). SEM programming can test typical models, but it can evaluate complex connections and models, such as the review of confirmation factors and the review of time series [160] [161]. Several terms should be familiar when operating the SEM:

- Observed and latent variables: The observed variables are immediately observable, for instance, number of methods line of code, and number of methods. The latent variable does not measure directly, such as software quality and software efficiency; therefore, the researcher uses the observed variable to measure latent variables. These measurements will be collected from coded answers and scientific experiments, and then calculate the score of these variables to show the value of the observations for the latent variables.
- Endogenous and exogenous latent variables: As a result, endogenous and exogenous latent variables are synonymous with dependant and independent variables. Two kinds of factor analysis exist, Factor analysis and confirmation factor.
- Factor analysis: The exploratory factor deals with unknown relations between the observed variable and the latent variable. While the confirmation factor is used when the relationship is clear between the observers to the latent variable [162].

6.3 IBM SPSS AMOS Evaluation Tool

The AMOS is an auxiliary state demonstration (SEM) visual program. The AMOS is measurable programming, which represents the study of moment structures [163]. AMOS is using SPSS module and is particularly used to model structural equations, examine pathways and study corroborating factors. It is used to review covariance or causal display programming. In AMOS, we can draw graphic templates using basic drawing peripherals. Chart calculations can be created quickly and statistically represent the results in several types of graphics [164]. AMOS GUI is very simple to use. It can design a specific model and determine the attributes and then import the extracted data. The numerical methods run in AMOS are among the best and most robust accessible. Researchers have used AMOS in SEM to evaluate quantitative data in many fields of research.

6.3.1 Amos Software properties

The IBM AMOS software has the capabilities of various kinds of design diagrams. The theses diagram is a visual representation of particular variables of various types of hypothetical models. Attributes can be represented with geometric shapes, for instance, A rectangle corresponds to the observed variable. An Oval can represent a latent variable. The single-point arrow is the effect of a variable on another variable. The double-headed arrow is the correlation between two variables, and the circle is the potential errors in the variable dataset. Table 6.1 below is presents these representations of the symbols [164].

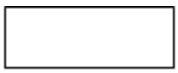
Name	Symbol	Explanation
Measured Variable		Variables that measured directly can be represented by rectangle
Latent Variable		Variable (latent) that cannot be measured directly and require several measured variables to estimate it can be represented by An Oval
Error variable		Error can be represented with circle shape
Causal effect (Direct effect)		The direct effect between variables can be represent by single arrow path
Covariance (none-directional path)		Covariance relationship between variables can be represented by curved two arrows.

Table 6.1 AMOS symbols model representation

6.4 Hybrid application system Product Quality Models in AMOS model:

AMOS has been used to design hybrid quality measurement model. This model has been statistically analysed and assess using extracted data from five open-source projects. The data set for each project was gathered using a java program and Code-MR plug-in. There are several metrics have been examined in the model, such as (extensibility, complexity, service load behaviours, code weaving and reusability) which have been represented as latent variable eclipse forms, as shown in 6.1. The sub-characteristics of each measurement were represented by a square or rectangular (latent) measurement variable that can be measured directly. Each sub-characteristic is abbreviated

to represent these sub-characteristics, as described in Table 6.2 below, the procedure of evaluating and designing the quality model.

Variable name	Description	Variable Type
Complexity	Hybrid Software Quality AO OO	Latent Variable
Extensibility	Hybrid Software Quality AO OO	Latent Variable
Service loading behavior	Hybrid Software Quality AO OO	Latent Variable
Code Weaving	Hybrid Software Quality AO OO	Latent Variable
Reusability	Hybrid Software Quality AO OO	Latent Variable
Ex1 ,Ex2, Ex3,Ex4	Extensibility	Measured Variable
Comp1, Comp 2, Comp 3 , Comp 4, Comp 5	Complexity	Measured Variable
SLB1,SLB2, SLB3, SLB4,SLB5	Service loading behaviour	Measured Variable
RU1, RU2, RU3, RU4, RU5	Re usability	Measured Variable
CW1, CW2, CW3, CW4	Code Weaving	Measured Variable

Table 6.2 latent variable representation in Hybrid Quality Model in AMOS

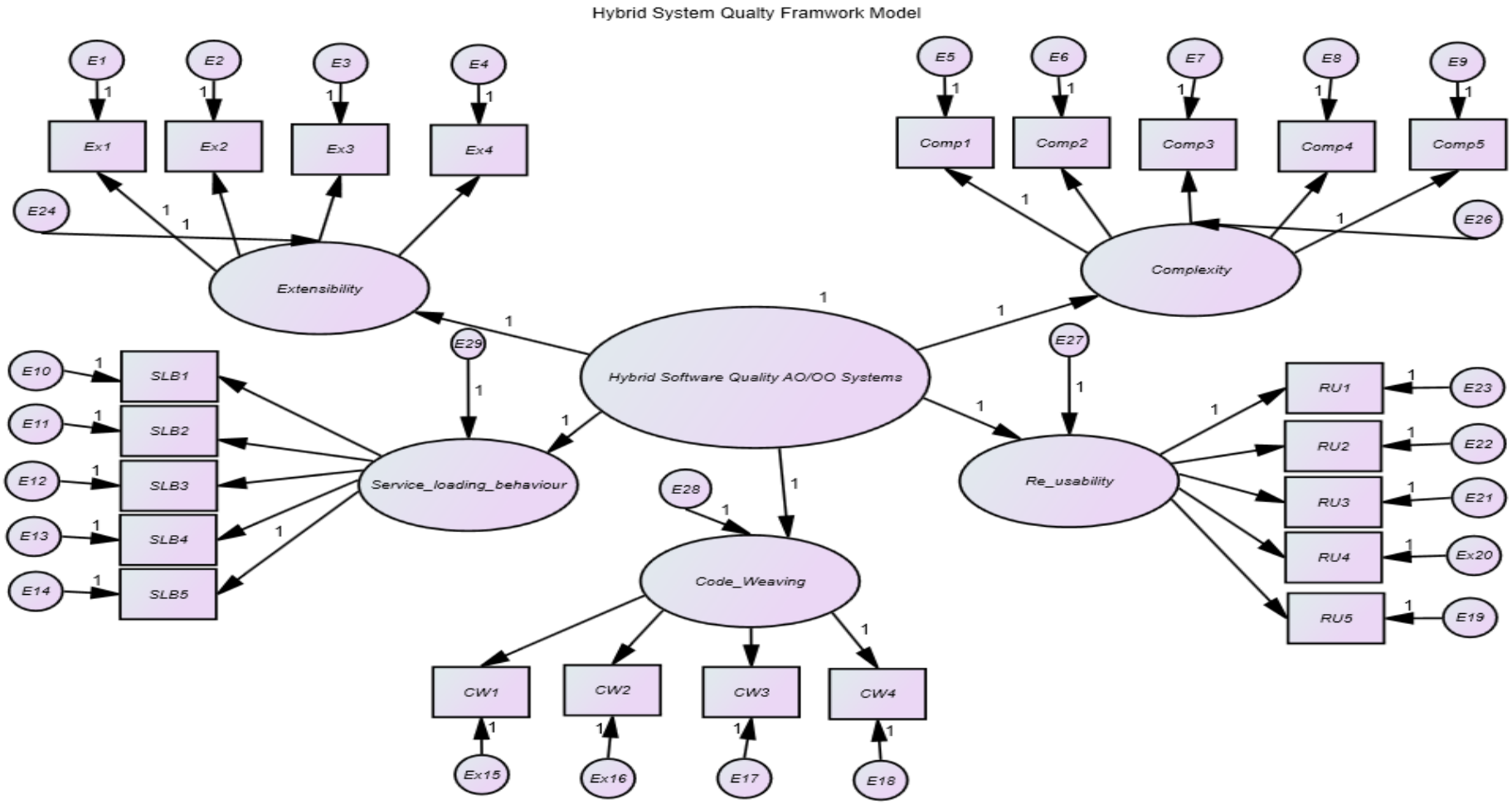


Figure 11 AMOS Diagram for Hybrid System Quality Framework Model

6.5 Hybrid Application Quality Model AOP/OOP design:

The design of a hybrid application quality model in IBM AMOS shall identify the latent variables and observed variables listed in Table 32. As it appears each components have been categorized into specific metrics with specifics abbreviation. This technique allows simplicity and easy management of the design model.

Proposing Hybrid Quality Model AO/OOP Components	Metrics Extractions sub-characteristics of the Hybrid Quality Model AO/OOP	IBM AMOS Representation
Measured Variable	Latent Variable	Latent Variable Abbreviation
Extensibility	Crosscutting Degree of an Aspect	Ex1
	Joinpoints Extension	Ex2
	Concern Diffusion over Components	Ex3
	Number of Models	Ex4
Complexity	Weighted Method Count	COMP1
	Depth of Inheritance Tree	COMP2
	RFC Response For a Class	COMP3
	Responses for Aspect	COMP4
	Number Of Aspects	COMP5
Service Loading Behaviour	Weighted Joinpoints Counts	SLB1
	LOC Line Of Code class	SLB2
	Number of Methods	SLB3
	Number of Fields	SLB4
	LOC Line Of Code Aspect	SLB5
Code Weaving	Number of Children	CW1
	Coupling Between Object Classes	CW2
	Access to Foreign Data	CW3
	Coupling on Advice Execution	CW4
Reusability	Lack of Cohesion of Methods	RU1
	Lack of Cohesion Among Methods	RU2
	Lack Of Tight Class Cohesion	RU3
	Lack of Concern based Cohesion	RU4
	Lack of cohesion in Joint Points	RU5

Table 6.3 Hybrid Applications Model Representation in IBM AMOS

6.6 Data extraction:

Software applications share a common, manageable set of functionalities. This function will meet the user's needs in a particular program. A feature represents distinct aspects visible to the

user or quality, or characteristics of the system [159]. There are two types of characteristic viability, supported by some applications and commonalities available across all products. A model features contain both the commonalities and the variability caused by software-industry needs. In addition, the software documentation may be used in the re-engineering process. However, there is some lack of updating of these documents, as they did not update the documents when the product was updated and changed. Some products also do not provide documentation on software or specification templates. Additional costs related to the provision of these documents were identified. This research used various methods to extract the AOP/OOP quality measures separately and then combine them into the AOP/OOP hybrid application quality model. There are two types of measurements that have been taken from the AOP and OOP programming. IBM AMOS is a statistical tool used to assess the reliability of data collection. IBM AMOS accepts the SPSS file type and computes the results. The five projects have been tested using one model regarding the difference in size, design, and implementation method.

6.7 Quantitative Evaluation:

6.7.1 Statistical Mean

The statistical average is a particular mathematical average that is extremely helpful in software engineering and computing. Essentially, the statistical mean is a cycle of arithmetic means in that it includes all numbers in a dataset and divided by total number of the dataset. The arithmetic or statistical means have generally been used throughout the modern period and in computer programming time. Generally, if we have values X1, X2, Xn, the mean given by following equation [165]:

EQ 6.1

$$\overline{x} = \sum_{i} f_i m_i / n$$

The projects' data have been normalizing from 0 to 10, and then calculate all statistical operations using IBM AMOS. The table 6.4 below shows the mean values for each category in the projects.

Hybrid Quality Model AO/OOP Components	**Projects Name**	**Mean Values**
Extensibility	GtalkWAP	6.83
Complexity	GtalkWAP	5.324
Service loading behaviour	GtalkWAP	6.208
Code Weaving	GtalkWAP	5.005
Re-usability	GtalkWAP	4.13
Extensibility	AjHotdraw	6.63
Complexity	AjHotdraw	6.69
Service loading behaviour	AjHotdraw	6.28
Code Weaving	AjHotdraw	5.72
Re-usability	AjHotdraw	6.22
Extensibility	Health Watcher	5.28
Complexity	Health Watcher	5.95
Service loading behaviour	Health Watcher	6.60
Code Weaving	Health Watcher	6.14
Re-usability	Health Watcher	5.70
Extensibility	AjHSQLDB	6.13
Complexity	AjHSQLDB	6.67
Service loading behaviour	AjHSQLDB	7.57
Code Weaving	AjHSQLDB	6.58
Re-usability	AjHSQLDB	4.95
Extensibility	Contract4J5	7.066667
Complexity	Contract4J5	5.686667
Service loading behaviour	Contract4J5	5.773333
Code Weaving	Contract4J5	7.241667
Re-usability	Contract4J5	7.293333

Table 6.4 Mean Values for five open source projects for five components of AOP/OOP Hybrid Application Quality Model

The mean is a model of the information that has been collected. Its value is generally normal. Despite this the average is not routinely one of the real qualities in informational index. However, one of its important properties is that it limits mistakes in forecasting any incentive in the collected data. A significant property of the mean is that it includes each value of extracted data which determine the feature of specific metrics. Similarly, the average is the principal proportion of focal propensity in which the sum of the deviations of each value from the average is systematically zero. The results have showed that the AO/OO hybrid application quality model for five projects

typically has accepted values for value distribution. The maximum value is 7.29 and the minimum value is 4.95, these results indicate that data collection have acceptable means values.

6.7.2 Standard Deviation

The standard deviation is a proportion of the measurement of the variation or spread of a set of values. A small standard deviation shows that the qualities will be close to the average (also called normal estimation of the whole). In contrast, a high expectation gap shows the qualities are distributed over a wider range. Standard deviation is abbreviated by SD and is most spoken to in mathematical text and conditions by the lowercase Greek letter sigma σ, for the population standard deviation, or the Latin letters, for the sample standard deviation. The standard deviation is used to quantify trust for evidence. For example, the margin of error in survey information is dictated by calculating the normal standard deviation in the results if a similar survey was conducted on different occasions. This induction of a standard deviation is often referred to as an "estimate standard error" or "standard error" with an average. It is represented as the standard deviation of the multitude of recorded understandings of this population. If an infinite number of tests were performed and an average of each sample was processed, all these values were extracted using IBM AMOS. Based on [165], the standard deviation value is less than or equal to 1 that which becomes an acceptable value. As shown in Table 6.5, the SD for hybrid quality model components is accepted for AjHotdraw, HealthWatcher, and some components are accepted for GtalkWAP, AjHSQLDB Contract4J5 projects. It was partially accepted concerning their standard deviation value over one or less than one. Therefore, the standard deviation or standard error values for the extracted hybrid quality model for the AO/OOP model measurements are accepted for data extraction for the proposed specific measures.

.

Hybrid Quality Model AO/OOP Components	Projects Name	Standard Deviation
Extensibility	GtalkWAP	1.0302
Complexity	GtalkWAP	0.7524
Service loading behaviour	GtalkWAP	1.2972
Code Weaving	GtalkWAP	1.419
Re-usability	GtalkWAP	1.6654
Extensibility	AjHotdraw	0.711
Complexity	AjHotdraw	1.047
Service loading behaviour	AjHotdraw	1.195
Code Weaving	AjHotdraw	1.05
Re-usability	AjHotdraw	1.149
Extensibility	Health Watcher	0.904
Complexity	Health Watcher	0.863
Service loading behaviour	Health Watcher	0.917
Code Weaving	Health Watcher	1.06
Re-usability	Health Watcher	0.962
Extensibility	AjHSQLDB	0.564
Complexity	AjHSQLDB	0.539
Service loading behaviour	AjHSQLDB	0.721
Code Weaving	AjHSQLDB	0.633
Re-usability	AjHSQLDB	0.586
Extensibility	Contract4J5	0.522338
Complexity	Contract4J5	0.522115
Service loading behaviour	Contract4J5	0.698437
Code Weaving	Contract4J5	0.506319
Re-usability	Contract4J5	0.582743

Table 6.5 Standard Devotion Values for five open source projects for five components

6.7.3 Factor Analysis:

Factorial analysis is a mathematical technique that provides information on the reliability, quality, and validity of data. The overall aim of factor analysis is to understand what expands the values or the value of the items. These values are used to evaluate or show the hypothesis or part of a particular model to be tested. It also breaks down covariate among the elements into significant components. Higher correlations for each element are resulting in higher internal reliability of values between elements. Factorial analysis may not be measured with a single element. It takes

three or more things. A higher correlation between the elements together with the factor needs to be measured.

.

Two kinds of factorial analysis exist.

- Exploratory Factor Analysis (EFA)
- Confirmatory Factor Analysis (CFA)

The major difference between EFA and CFA is that EFA focuses on the number of elements found, not the relationships between the elements. This research proposes a hybrid model of application quality and emphasises CFA to test and evaluate the values of each component of the model. CFA is looking at the extent to which the hypothetical factorial structure is consistent with the model. The change of the model shows a mathematical comparison of the correlations, i.e., the covariance between the elements concerning the correlations expected by the model under test. Mathematically, some models involve certain relationships. For example, if the model shows that two factors are not correlated, the model is not appropriate because an element will be highly correlated with another factor. If the model has high values of correlated factor, the uncorrelated elements will lead to a wrong fit to the model. However, many load factors can be specified by the user for the model, and any item that can be loaded on multiple factors is desirable [166] [167]. Researchers have been worked on many fit indicators such as Chi-Square; $\chi 2$ lower values mean better fit. RMSEA's lower value (< 0.06) value showed the best results. SRMR, lower values show excellent results for factors fit (<0.08). Comparative fit index and Tucker Lewis index, higher values show better fit (>0.95) [166] [167]. The acceptable values of factor loading are>0.5 [217]. The five project load factors, most of them are over 0.5. This value shows that the extracted metric values are adjusted to the proposed quality model and reliable. Table 6.6 presents the details of the factor analysis and values for selected projects.

Metrics Characteristics	Projects	Factor Loading values
Complexity	GTalkWap	0.8
Extensibility	GTalkWap	0.5
Service loading behaviour	GTalkWap	0.5
Code Weaving	GTalkWap	1.2
Re_usability	GTalkWap	0.7
Complexity	AjHotdraw	1.1
Extensibility	AjHotdraw	0.9
Service loading behaviour	AjHotdraw	1
Code Weaving	AjHotdraw	0.9
Re_usability	AjHotdraw	1
Complexity	Healthwatcher	0.2
Extensibility	Healthwatcher	0.9
Service loading behaviour	Healthwatcher	1.1
Code Weaving	Healthwatcher	1
Re_usability	Healthwatcher	1
Complexity	Ajhsqldb	1
Extensibility	Ajhsqldb	1
Service loading behaviour	Ajhsqldb	0
Code Weaving	Ajhsqldb	0
Re_usability	Ajhsqldb	1
Complexity	Contract4J5	1
Extensibility	Contract4J5	1
Service loading behaviour	Contract4J5	0.5
Code Weaving	Contract4J5	0.9
Re_usability	Contract4J5	1

Table 6.6 Factor Loading values for all projects

6.7.4 Significance P value

The p-value is the probability of getting results from any case as outrageous as the observed side effects of a statistical hypothesis test, assuming that the invalid hypothesis is correct. The p-value is used as an option versus termination, giving the least centrality to which the invalid hypothesis would be rejected. A more diffident p-value implies that there is more grounded proof of the elective theory. The invalid hypothesis expresses no link between the two factors considered (one variable has no influence on the other). It expresses the results are because of the possibility and are not enormous in terms of the support of the idea under consideration. So the invalid assumption is that what you're trying to prove has not happened. The level of statistical significance is regularly reported as a p-value between 0 and 1. The most modest estimate of p, the most founded evidence that you should reject the invalid assumption. A value of less than 0.05 (usually 0.05) is statistically significant. It shows firm evidence against the invalid theory, because there is

not exactly a five percentage (5%) probability the invalid is just (and the results are irregular). Second, we reject the invalid hypothesis and recognise the optional hypothesis. A p-value greater than 0.05 (> 0.05) is not statistically significant and shows robust evidence of the null hypothesis. They show acceptance of the null hypothesis and rejection of an alternate hypothesis. Table 6.7 demonstrate all the p-values for selected project [168].

Projects	Ex1	Ex2	Ex3	Ex4	Comp 1	Comp 2	Comp 3	Comp 4	Comp 5	SLB 1	SLB 2	SLB 3	SLB 4	SLB 5	RU1	RU2	RU3	RU4	RU5	CW 1	CW 2	CW 3	CW 4
Quality Metrics	Extensibility				Complexity					Service loading behaviour					Reusability					Code Weaving			
GTalkWap	0.001	0.001	0.001	0.001	0.3	0.3	***	***	***	***	0.001	***	***	0.001	0.001	***	***	***	***	***	***	***	0.001
AjHotdraw	0.001	***	***	0.7	***	0.7	***	0.4	***	***	***	***	***	0.001	000.1	0.8	0.3	***	0.001	0.1	0.3	***	0.001
Healthwatcher	0.001	***	***	***	0.001	***	***	***	***	0.001	***	0.001	***	0.001	0.001	***	***	0.4	0.001	***	***	0.2	0.001
Ajhsqldb	0.001	***	***	***	0.001	0.001	0.001	0.001	0.001	0.001	***	***	***	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Contract4J5	0.001	***	***	***	0.001	0.3	***	***	***	0.2	0.1	0.1	0.6	0.001	0.001	0.001	***	0.001	***	0.1	***	***	0.001

Table 6.7 Significant P value for 5 projects

The significant P-value was computed and extracted in the five open source projects, as shown in Table 36. In the GtalTalk project, the Extensibility metrics have the value of 0.001 of the p-value. This means that there are strong relationships between sub-characteristic measurements and extensibility characteristics in the GtalTalk hybrid application model. All other p-values also show strong relationships between the sub-characteristic measurements and the quality model components. P-values have been showed by three asterisks ***, which is stands for over 0.05, which is statistically significant. In project AjHotdraw, Healthwatcher, and Contract4J5, all the software quality components and sub characteristics have significant p values, which show a statistically significant value to the effect relations of these components in the hybrid application model. The Ajhsqldb project has statistically non-significant p-values in the complexity of three

Components re-use and code weaving. On average, all projects were tested against p-values. They have been showed the relationships and effects between the sub-characteristic measures and the components of the product quality model for hybrid application systems.

6.7.5 Correlation Coefficient

Correlation is a measurable strategy used to explore a potential direct opportunity between two continuous factors. It is essential to verify and interpret at the same time. Correlation is a technique to study a potential bidirectional direct relationship between two constant factors. The relationships were estimated by measuring the correlation coefficient, which shows the strength of the suspected direct relationship between the factors mentioned. Correlation is a method to evaluate a potential two-way linear association between two continuous variables. The correlation is measured by a statistic called a correlation coefficient, representing the strength of the assumed linear association between the variables involved. This is a dimension-free quantity which takes a value in the range from -1 to +1. A correlation coefficient of zero shows that there is no linear relation between two continuous variables, and a correlation coefficient of -1 or +1 shows a perfect linear relation. The strength of the relation can be from -1 to +1. The higher the correlation, the closer it gets to 1. If the coefficient is a positive number, the variables are directly related (meaning that the value of a variable increases, the other tends go up). A coefficient is a negative number, the variables are inversely related (i.e., as the value of one variable goes up, the value of the other goes down). Any other relationship between two continuous variables that is not linear is not statistically correlated. To underline this point, a mathematical relationship is not necessarily a correlation [168]. Pearson's item second correlation coefficient is meant as ρ for a populace boundary and as for an example measurement. It is used when both factors are normally communicated. This coefficient is influenced by extraordinary qualities, which may overstate or pipe the strength of relationship, and is consequently improper when either of the two factors is not regularly circulated [169]. For a correlation between the x and y factors, the recipe for calculating the Pearson tie-in coefficient example is given by

:

EQ 6.2..... $r = (n (\sum xy)- (\sum x)(\sum y))/(\sqrt{} [n \sum x^2-(\sum x)^2][n \sum y^2– (\sum y)^2)$

Projects name	Minimum value of Correlation	Maximum value of correlations	No. of Item
GTalkWap	-0.873	0.893	23
AjHotdraw	-.391	.532	23
Healthwatcher	-.588	.445	23
Ajhsqldb	-.548	.581	23
Contract4J5	-.668	.535	23

Table 6.8 Correlation coefficient for 23 items for 5 projects

As shown in Table 6.8, the correlation coefficient of 23 sub-features is significantly correlated with perfectly linear relations. These strong relationships indicate important relationships between these elements of the hybrid software quality model. All five projects showed an acceptable value for the correlation factor, which is statistically significant.

6.7.6 Average Variance Extracted and Construct Reliability

Average Variance Extracted AVE is a proportion of the measure of change that is taken by a development as a function of variance because of an estimation error. A measurement is used to estimate convergent validity. Similar to the clarified difference in the EFA's exploratory factor analysis, the AVE is the average variance of the pointer factors that a construction seeks to clarify. The AVE for each construction can be got by the number of squares of fully normalised factor loads isolated from that total and adding up the error deviations for the indicators. For the standardised arrangement, all cursor and latent factors are scaled to have a unit variance. The AVE is >= 0.5, which considers an accepted value for data validity between the corresponding variable in template [171] [172]. The extracted mean variance may be computed using the specific formula

.

EQ 6.3..... $$AVE = \frac{\sum_{i=1}^{k} \delta_i^2 \text{---}}{\sum_{i=1}^{k} \delta_i^2 \text{---} + \sum_{i=1}^{k} Var\ e_i \text{---}}$$

Where δ is number of items, δi the factor loading of items i and (Var ei) the variance of error item i.

Composite reliability (building reliability) The CR is a measure of the inner coherence of the elements of the scale, much like the Cronbach alpha [170]. This may very well be considered the equivalent of the aggregate sum of the actual variance of the scores compared with the total variance of the scores on the scale [171]. Again, it is an "indicator of the variation divided between the observed factors used as a pointer of a latent construction" [171].

The reliability thresholds for composites are questionable (an acceptable value can range from 0.60 to more), and various authors offer various threshold recommendations. Many of them are based on how many things you have on your scale. Small amounts of scale things will generally lead to lower levels of reliability, while larger amounts of scale things will generally have more significant levels [172]. The formula of CR as the following:

Formula

$$\boldsymbol{EQ\ 6.4} \ldots\ldots\ \boldsymbol{CR} = \frac{(\sum_{i=1}^{p} \delta_i)^2}{(\sum_{i=1}^{p} \delta_i^2)^2 + \sum_{i}^{p} \mathrm{V}(\delta \mathrm{i})}$$

i = standardized loading indicator,

V (δi) = variance of the error for the indicator,

p = number of indicators

Projects	Ex 1	Ex 2	Ex 3	Ex 4	Comp 1	Comp 2	Comp 3	Comp 4	Comp 5	SLB 1	SLB 2	SLB 3	SLB 4	SLB 5	RU 1	RU 2	RU 3	RU 4	RU 5	CW 1	CW 2	CW 3	CW 4
Quality Metrics	Extensibility				Complexity					Service loading behaviour					Reusability					Code Weaving			
AVE (GTalkWap)	0.317				0.46					0.746					0.525					0.655			
CR (GTalkWap)	0.731				0.729					0.559					0.703					0.579			
AVE(AjHotdraw)	0.470				0.356					0.330					0.392					0.4125			
CR(AjHotdraw)	0.679				0.763					0.770					0.752					0.701			
AVE(Healthwatcher)	0.570				2.74					0.444					0.280					0.310			
CR(Healthwatcher)	0.632				1.129					0.735					0.782					0.734			
AVE(Ajhsqldb)	0.600				0.250					0.480					0.250					0.367			
CR(Ajhsqldb)	0.750				0.722					1.000					0.559					0.716			
AVE(Contract4J5)	0.777				0.666					0.410					0.390					0.389			
CR(Contract4J5)	0.470				0.629					0.746					0.753					0.710			

Table 6.9 Average Variance Extracted and Construct Reliability

Table 6.9 explained the Average Variance Extracted AVE values. The AVE threshold is >=0.5, which is an accepted value. The minimum number is 0.250 and the maximum number is 0.777. Out of 5 components of the hybrid application model, three components have acceptable AVE values. This shows the validity of the data between the corresponding variables in the model that proposed 23 sub-characteristics with the five general components is acceptable. Construct Reliability RC build was extracted and showed in Table 38. The CR values show the internal coherence of the scale items for all components of the hybrid application model. The minimal value of CR 0.470 and the maximum value was 1.129. These CR results show the reliability of the composite rating of the component in the proposed that has a value of 0.6 or above is acceptable. CR showed that the average values of the five components for proposed hybrid application quality model are reliable and qualifying.

6.7.7 Cronbach's alpha and Kaiser-Meyer-Olkin (KMO) Test:

Cronbach's alpha is a proportion of internal coherence: how closely related elements are in groups. It is considered a measurement of the reliability of the scale. A "high" value for alpha does not suggest that it is one-dimensional. If, despite the internal consistency, you want to prove that the scale is one-dimensional and can be performed additional studies. The exploration factor study is a dimensional verification technique. Technically, Cronbach's alpha is definitely not an objective test it is a coefficient of reliability (or coherence).

The Formula of Cronbach's alpha as the following:

$$EQ\ 6.5 \ldots\ldots \alpha = \frac{N\overline{c}}{\overline{v} + (N-1)\overline{c}}$$

A Kaiser-Meyer-Olkin (KMO) test is used in research to determine the appropriateness of inspecting the information to be used for factor analysis. In this a measure that shows how much fluctuation in your factors can be caused by fundamental factors. High qualities (near 1.0) by and large demonstrate that a factor examination might be valuable with your information. If the KMO value is below 0.50, the impact of factor analysis will likely not be useful [173][174][175].

The Formula of as following

$$\text{EQ 6.6} \ldots\ldots \text{MO}_{\text{j}} = \frac{\sum_{\text{i}\neq\text{j}} \text{r}_{\text{ij}}^2 \text{ -}}{\sum \text{i} \neq \text{j}\ \text{r}_{\text{ij}}^2 + \sum \text{i} \neq \text{j}\ \text{u}}$$

Where:

MOj is the test outcome.

• rij 2 is the correlation matrix.

• uij is the partial covariance matrix.

Projects	EX1	Ex2	Ex3	Ex4	Comp1	Comp2	Comp3	Comp4	Comp5	SLB1	SLB2	SLB3	SLB4	SLB5	RU1	RU2	RU3	RU4	RU5	CW1	CW2	CW3	CW4
Quality Metrics	Extensibility				Complexity					Service loading behaviour					Reusability					Code Weaving			
CA (GTalkWap)	0.708				-0.588					0.346					-0.071					-0.48			
KMO(GTalkWap)	0.702				0.517					0.418					0.474					0.508			
CA(AjHotdraw)	0.414				0.241					0.387					0.008					0.174			
KMO(AjHotdraw)	0.496				0.486					0.516					0.510					0.420			
CA(Healthwatcher)	0.414				-0.546					0.074					0.204					-0.149			
KMO(Healthwatcher)	0.533				0.448					0.438					0.390					0.545			
CA(Ajhsqldb)	0.655				-0.173					0.602					-0.674					-0.155			
KMO(Ajhsqldb)	0.681				0.530					0.590					0.418					0.605			
CA(Contract4J5)	0.708				-0.499					0.363					-0.453					-0.015			
KMO(Contract4J5)	0.533				0.448					0.438					0.390					0.545			

Table6.10 Five open source projects Cronbach's alpha and Kaiser-Meyer-Olkin (KMO) Test values

Table 6.10 explains Cronbach's alpha and Kaiser-Meyer-Olkin (KMO) values for five open source projects show the reliability and validity of these values. Cranach's alpha (CA) minimum value is -0.015, and the maximum value is 0.708. The CA values indicate that there is no meaningful consistency between the sub-characteristics and the major components of the proposed product quality model for hybrid application framework. The average values are below the limit value. Kaiser-Meyer-Olkin (KMO) minimum value is 0.390 and maximum values are 0.702. The average values of KMO are more than 0.5 which considers as acceptable values. These values suggest that all model data are suitable and reliable for all components of the proposed model. Moreover the results have showed how significant KMO values in measuring, sampling adequacy for each model variable.

6.8 Hybrid Application System Quality Model Discussion:

This section discusses the hypothesis of statistical evaluations that have been evaluated, the hybrid application model in section 6.4. There are five model assumptions that require proof. These hypotheses have been derived from the five main components of the model (Extensibility, Complexity, and Service loading behaviours, Code Weaving, and Re-usability).

Hypothesis 1 (Extensibility) H1: These will have an impact on the model's functionality. As discussed in Section 5.4, the three parameters affected scalability (CDA, JPE, CDC and NOM). Extensibility will affect the functionality of the quality model and extend the crosscutting degree, commonality, concern across the source project, and the number of models in highly reliable techniques. According to section 5.7, the extensibility hypotheses have been accepted. As shown in Table 6.11, H1 has been accepted in most statistical evaluations for Cronbach alpha and construction reliability almost achieved the threshold results. To conclude, H1 was approved and agreed.

Hypothesis1 (H1)	**Extensibility(EX)(Ex1,Ex2,Ex3,Ex4)**							
Statistical Evaluation	Statistical Mean	Standard Deviation	Significant P value	Correlation Coefficient	Average Variance Extraction	Cronbach Alpha	Kaiser Meyer Olkin	Construct Reliability
Threshold Values		<=1	>=0.05	-1 to +1	>=0.5	>=0.6	>=0.5	>=0.7
Obtained values Average 5 projects	6.387	0.746	0.065	Min 0.597 and Max - 0.613	0.546	0.579	0.589	0.652
Results	Accepted	Accepted	Accepted	Accepted	Accepted	Not Accepted	Accepted	Not accepted

Table 6.11 Hypothesis1 H1 statistical evaluation

Hypothesis 2 (Complexity) H2: These components will affect the reliability of the proposed hybrid quality model, as discussed in Section 5.4. The Complexity in the model has been

measured by five metrics (Weighted Method Count (WMC), Depth of Inheritance Tree (DIT), RFC Response for a Class (RFC), Responses for Aspect (RFA), Number of Aspects (NOA)). The complexity negatively affected reliability. When complexity is increased, the reliability of the system decreases. As shown in Table 6.12, H2 was accepted based on the statistical evaluation five of 7 factors were accepted. These results indicate that most projects were more complex and made the system less reliable.

Hypothesis2 (H2)	**Complexity(Comp)(Comp1,Comp3,Comp3,Comp4, Comp5)**							
Statistical Evaluation	Statistical Mean	Standard Deviation	Significant P value	Correlation Coefficient	Average Variance Extraction	Cronbach Alpha	Kaiser Meyer Olkin	Construct Reliability
Threshold Values	5	<=1	>=0.05	-1 to +1	>=0.5	>=0.6	>=0.5	>=0.7
Obtained values Average 5 projects	6.064	0.744	0.240	Min 0.597 and Max - 0.613	0.894	-0.313	0.4858	0.794
Results	Accepted	Accepted	Accepted	Accepted	Accepted	Not Accepted	Not Accepted	Accepted

Table 6.12 Hypothesis1 H2 statistical evaluation

Hypothesis 3 (Service Loading Behaviour) H3: These Components will affect the efficiency of the components in the quality model of hybrid application systems. Service Loading Behaviour has been measured by these metrics (Line of Code (LOC), Number of Methods (NOM), NOF (Number of Fields), Weighted Joint points Counts (WJC), Line of Code Aspect (LOCA)) as it shows in table 6.13.

Hypothesis3 (H3)	**Service Loading Behaviour (SLB)(SLB1,SLB2,SLB3,SLB4, SLB5)**							
Statistical Evaluation	Statistical Mean	Standard Deviation	Significant P value	Correlation Coefficient	Average Variance Extraction	ronbach Alpha	Kaiser Meyer Olkin	Construct Reliability
Threshold Values	5	<=1	>=0.05	-1 to +1	>=0.5	>=0.6	>=0.5	>=0.7
Obtained values Average 5 projects	6.486	0.965	0.302	Min 0.597 and Max - 0.613	0.482	0.354	0.48	0.762
Results	Accepted	Accepted	Accepted	Accepted	Accepted	Not Accepted	Accepted	Accepted

Table 6.13 Hypothesis1 H3 statistical evaluation

As shown in Table 42 above, all statistical measurement results indicate the acceptance of this hypothesis H3. The H3 indicates that service loading behaviour will positively affect the efficiency of our proposed hybrid quality model. As the SLB increases, the system becomes more efficient.

Hypothesis 4 (Code Weaving) H4: these components will affect Maintainability in the quality model of hybrid application systems. Code Weaving has been measured by these metrics (Number of Children (NOC), Coupling between Object Classes (CBO), and Access to Foreign Data (ATFD), coupling on Advice Execution (CAE)). Table 6.14 has been shown the acceptance of H4 is used for Code Weaving evaluation. The higher value of CW will result from the higher Maintainability of the system. The Maintainability of the system is helping a successful repair in a given time

Hypothesis4(H4)	**Code Weaving(CW1,CW2,CW3,CW4)**							
Statistical Evaluation	Statistical Mean	Standard Deviation	Significant P value	Correlation Coefficient	Average Variance Extraction	Cronbach Alpha	Kaiser Meyer Olkin	Construct Reliability
Threshold Values	5	<=1	>=0.05	-1 to +1	>=0.5	>=0.6	>=0.5	>=0.7
Obtained values Average 5 projects	6.137333	0.933	0.120	Min 0.597 and Max -0.613	0.426	-0.125	0.524	0.688
Results	Accepted	Accepted	Accepted	Accepted	Not Accepted	Not Accepted	Accepted	Accepted

Table 6.14 Hypothesis 4 (Code Weaving) statistical evaluation

Hypothesis 5 (Re-usability) H5: These components will influence the portability of the hybrid application system quality model. Reusability has been measured by these metrics Lack of Cohesion of Methods (LCOM), Lack of Cohesion among Methods (LCAM), Lack of Tight Class Cohesion (LTCC), Lack of Concern Based Cohesion (LCBC), and Lack of Cohesion in Joinpoints (LCJ). Table 44 explained the effect of Reusability on the Portability of quality mode. The higher value of reusability leads to an increased value of portability. Software Engineering Portability enables the application system to be moved from one environment to another at a lower cost. Adding more Portability is a high level of programming language with reusability. However, both of them measured the same software in different environments. The Table 6.15 is shown the re-usability partially accepted, which shows average values of Portability of the application systems.

Hypothesis 5 (H5)	Re-usability (RU1,RU2,RU3,RU4,RU5)							
Statistical Evaluation	Statistical Mean	Standard Deviation	Significant P value	Correlation Coefficient	Average Variance Extraction	Cronbach Alpha	Kaiser Meyer Olkin	Construct Reliability
Threshold Values	5	<=1	>=0.05	-1 to +1	>=0.5	>=0.6	>=0.5	>=0.7
Obtained values Average 5 projects	5.658	1.030167	0.3	Min 0.597 and Max - 0.613	0.367	0.197	0.436	0.709
Results	Accepted	Accepted	Accepted	Accepted	Not Accepted	Not Accepted	NotAccepted	Accepted

Table 6.15 Hypothesis 5 H5 statistical evaluation

6.8.1 Hypothesis Model (Software Product Quality-Hybrid Application System):

Section 6.5 was discussed in detail and all hypotheses were accepted using 7 statistical measures. Software Product Quality - Hybrid Application System Model has been tested and approved according to five assumptions (H1, H2, H3, H4, and H5). All of these hypotheses have been derived from four or five sub-characteristics (metrics) which represent measurements of quality. These sub-characteristics had no acceptable value, but 5 out of 7 average results will be accepted. The hypotheses model may be presented in Figure 6.2 [178] [179].

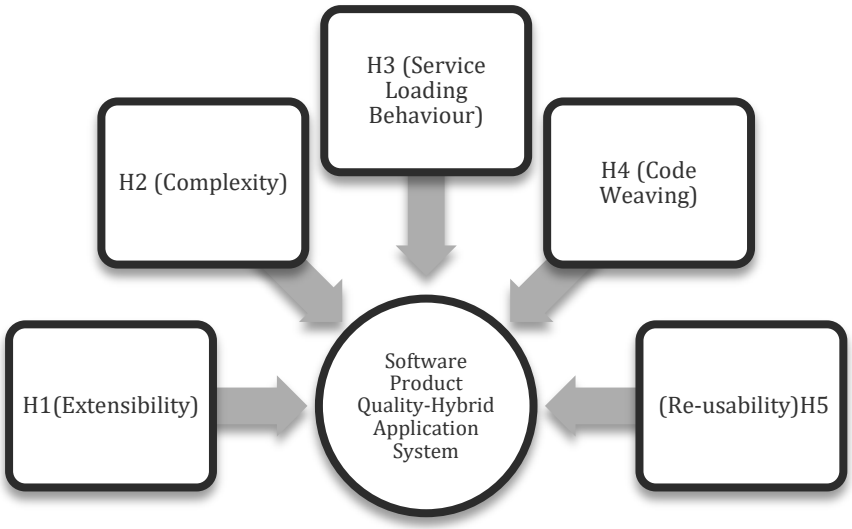


Figure 6.2 Hypothesis Model (Software Product Quality-Hybrid Application System)

6.9 Conclusion

This chapter concentrated on the evaluation of the proposed Software Product Quality - Hybrid application system model. Statistical assessment and validation methodologies were used to analyse data collected from five open projects. The statistical evaluation comprises (mean, standard deviation, factor analysis - P-value, correlation coefficient, mean variance extracted, construction reliability, Cronbach Alpha and Kaiser Mayer Olkin). Statistical evaluations have a threshold values for accepting and rejecting the extracted values for all measures (sub-components), which affect the major components of the model. The main proposed components (Extensibility, Complexity, Service Loading Behaviours, Code Weaving, and (Re-usability). These major components have been tested in a form of 5 hypotheses (H1, H2, H3, H4 and H5). All hypotheses have been approved. The statistical results for H1 (Extensibility) show that the higher value of extensibility increases the functionality of the proposed software product quality - Hybrid Application System. Extensibility of AOP has included crosscutting degree, joint point, and concern across hybrid application system. H2 (Complexity) has been accepted in the model. H2 affected by (Line Of Code (LOC), Number of Methods (NOM), NOF(Number of Fields), Weighted Joint points Counts (WJC), Line Of Code Aspect (LOCA)), however, H2 has been accepted, but more complexity of the system lead to poor reliability of the hybrid application systems. H3 (Service Loading Behaviours) entered Software Product Quality - Hybrid application System proposed model, as the statistical result shows. H3 was assigned to (Line of Code (LOC), Number of Methods (NOM), NOF (Number of Fields), Weighted Joint Points Counts (WJC),

Line of Code Aspect (LOCA)). The rise in the value of H3 indicated that the hybrid system will be more efficient. H4 (Code Weaving) in Product Quality – Hybrid Application System software was reviewed and evaluated. H4 affected by (Number of Children (NOC), Coupling between Object Classes (CBO), Access to Foreign Data (ATFD), coupling on Advice Execution (CAE)). H4 will enhance the maintainability of the proposed Software Product Quality - Hybrid application system. H5 (Re-usability) is the hypothesis that has been examined and statistically evaluated. A H5 result shows that it has been accepted. H5 affected by Lack of Cohesion Of Methods (LCOM), Lack of Cohesion Among Methods (LCAM), Lack of Tight Class Cohesion (LTCC), Lack of Concern Based Cohesion (LCBC), Lack of Cohesion in Joint Points (LCJ). H5 will affect the portability of the proposed model and help the hybrid application system to be more agile in different environments. In conclusion, all five assumptions were accepted, and the model was evaluated and accepted. The model shows that the proposed metrics and major components are accepted. The combination of AO/OO measurements was conducted and measured and then designed using IBM AMOS software. Results were collected and quantified statistically. The similarity of the measurement results shows that extraction of the measurements is correct and showed common measurements between these technologies that affect the different measurements. AOP measurements have a similar impact on the development process of OOP measurement software according to the got results [180][181].

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Introduction:

This chapter concludes the work carried out in this research. This research clarifies the methods of analysis of the hypothesis and the strategies that have been used to prove these hypotheses. The contribution to the knowledge has been explained, which is proposing new product software quality for a hybrid application system. Adding more, the study shows the relationships between AO and OO metrics with similarities of several metrics. The limitations and inadequacies of this study are that several case studies, size of the project and development methodologies, which need more analysis to get more results and get more accurate results. The chapter ends with a clarification of the future direction of this study. Section 7.3 describes the major contribution, design, and implementation of this research. Section 7.4 deals with the limitation of the study because of the duration and complexity of implementing this project. Section 7.5 discusses the future work that will need to be done in the future. Finally, section 7.6 concludes the importance of this research in the software development measurement method.

7.2 Achieved Goals and Objectives:

The main aim of this search was:

Propose a new Aspect Oriented quality measurement and a new software product quality model for hybrid application systems. In section 3.5, have been explained several tools are being tested for extraction AOP metrics. However, not all the purpose and objectives of this study have been achieved. It clarifies what is lacking in the development of AOP metrics and quality metrics of AOP products. Several tools have been used for extracting metrics from hybrid application, including Ajatoo, AOP Metrics, CMTJava, JDepend, JMetric, Rapid miner and GATE tools. The disadvantages of these tools that have issues in compatibility between AO/OO and most of them not recognise AO technology in the project source code. The results identified a research gap and the absence of evaluation measures for hybrid application software AO/OO. Proposing and approving a novel Aspect Oriented quality model for measuring hybrid application product. Object Oriented Programming and Aspect Oriented Programming have different quality metrics, but there are no existing metrics that measure a hybrid application system (AOP-OOP). Chapter 5, Section 5.3 it has been discussed five types of quality models. ISO/IEC 9126 quality measurement model has been adapted to propose a new product quality model.

A comprehensive literature review and finding of the research gap between AO and OO metrics for the hybrid application have been conducted which have been explained in Chapters 2 and 3. Five open source projects have been assessed and categorised to match the criteria of aim and aim of this thesis. However, intentionally it has selected these projects that have a difference in use AOP and OOP, in order to have accurate and reliable results.

The ISO/IEC 9126 software quality model has been analysed and examined qualitatively and quantitatively. It was proposed to add five key components to the model (Scalability, Complexity, Service Loading Behaviours, Code Weaving, and Reusability). Structural Equation Modelling was used to design this model and all key elements of the proposed model. These are represented as an observable variable in the proposed model. The principal components of the product Quality Model are (Functionality, Reliability, Effectiveness, Maintainability, Portability) respectively. These components have 23 sub-components. These have been extracted using the Code-MR plug-in to gather the values of these sub-components. The results were analysed and standardised to reflect each value in Chapter 4.

All these components have been approved and accepted in detailed literature reviews and statically using structure equation modelling using the AMOS tool.

The AMOS tool is used SPSS statistical function for validating data. 23 measurements were got using the Code-MR plug-in. This is an acceptable contributes to the quality model of the hybrid application product. These have been examined statistically. AMOS represents the proposed model using Graphical User Interface features. Comprehensive assessment has been conducted in Chapter 6 by using seven statistical measures to prove the reliability and acceptability of the extracted data. The collected data have been entered the IBM AMOS tool. IBM AMOS has SPSS statistics measurement capabilities. Five hypotheses have been proposed and proven (H1, H2, H3, H4 and H5). Static measurements have been carried out and all hypotheses are accepted. However, Complexity metrics have been accepted, but it has reduced the reliability of the hybrid application system.

7.3 Contribution:

There are a variety of quality models for measuring hybrid application available in the industry. However, neither of these models has been received as a standard strategy for the moment. As described in this thesis's motivations in Chapter 1 and Chapter 4, there is still space for novel approaches that could provide a product quality approach and comprehensively hybrid application developed using AOP and OOP.

Below are several contributions from this research:

The research analysed all existing Aspect Oriented approaches and the purpose Hybrid AO/OO quality framework (some of the well-known approaches were explained and criticised in Chapter 3). It was observed that there are still limitations to the existing software quality model of hybrid application system approaches (AO, OOP). The limitation was the existing quality model could not estimate the effect of AO/OO metrics on software quality, while proposed metrics have been identified the relationship between quality metrics for hybrid AO/OO software applications.

Proposing a software quality software model for measuring hybrid AO/OO software application based on gap in details literature review and ISO 9126 quality model.

Another contribution is proposing a model for evaluating software quality model for hybrid application systems. It has been used tool to design and evaluate the model, which is Standardised Equations Modelling. As a result, research has focused on developing a unified approach for Product Quality Framework of Hybrid Application System.

The study has addressed quality measurement problems in hybrid application systems (AOP, OOP). which not give accurate measurement when used AO/OO technologies in their development process, while the proposing model measure all metrics values for both AO/OO precisely .

Finding the relationships between OO and AO by reflecting traditional OO metrics on AO metrics and unified it. The quality model contains components representing the relations between the principal components of the product and their sub-components, for instance, aspectual concern, such as aspect, Advice, and Pointcuts, JoinPoint, and weaving association combination with traditional OOP quality measurements metrics. Ratings described a new proposed quality model based on several quality frameworks, which were described in Chapter 4. New product quality metrics focused on the hybrid application system aspect were proposed in Section 5.9.

Propose product quality metrics for a hybrid application system. It must extract the values from the metrics. The Code-MR tool was used to extract the measurements with an extra written algorithm to extract the AOP and OOP measurements. All extracted data inserted into the SPSS file were then used as input data in the structural equation modelling. These measures were assessed and tested in a logical and statistical manner. The proposed parameters were accepted regarding threshold values for assessment methods. Sections 2.7 and 4.3 have explained these metrics.

Proposing new design for all components of product quality model using Structural Equation Modelling SEM and SPSS statistical tool is used to measure outcomes. Seven statistics were used to verify the reliability and acceptance of the quality model. Five hypotheses have been examined with the reflection of the model of quality. These assumptions were endorsed and accepted regarding the reliability of static values.

A comprehensive literature review was carried out. Several open-source code tools have been used to extract aspect-oriented and object-oriented measurements. Static and dynamic measurements were gathered. This study focused on static measurements because those measurements are based on the source code which can measure statically. The dynamic measures required the execution of the applications, which require more resources and more time for evaluation. Extraction of static measurements was used to determine which measurements affect the product quality model for the hybrid application system. However, dynamic metrics also extracted base on statistical measurements.

Several types of graphic representation of the static and dynamic measures were revealed. Code-MR is displays measurement results in graph output and table representation. It can easily filter and separate the result from the metrics and can also be retrieved as text, then feed into the proposed product quality framework model.

7.4 Study Limitation:

The implementation and assessment process for this study included several limitations:

At the beginning of the extraction of the Aspect measures, many tools were used. These tools have been developed to work with specific versions of AOP, and it was difficult to construct all of them, unifying the programming language of all projects to identify new AOP metrics.

The five open source projects have been developed using AOP and OOP programming techniques. The project's size varied. This affected the retrieval of the measurements and the results achieved. In some projects, AOP is more used than OOP and in others was opposite and the development method was varied.

The crosscutting concerns have been measured dynamically at the runtime. The difficulty of measuring all crosscutting concerns, Advice, Pointcut is calculating statically, not at runtime.

Configure multiple tools to extract AOP metric and OOP metric. These tools run on several versions of Java Development Kids. Different versions lead to different outcomes. It takes a

lot of time to normalise and conclude these outcomes. Combinations of standardised and unified AO and OOP measures have been collected.

No compositional framework exists to measure the quality of appearance and object as a quality measurement model.

7.5 Future Work:

Product quality frameworks for hybrid application system are a developing approach. There is still a gap for the improvement and extension. As in Section 7.3, a lot of work needs to be done on the software quality framework for hybrid application software due to time constraints and implementation complexity. Many additional components may be added to the quality framework template. Extension tools can be developed to measure the hybrid application system and have shown the effect of appearance and object measurements on every application. Aspect Oriented metrics require more research; many common Aspect and purpose measures need to be identified. Data extraction tools for identified new metrics can be implemented to help software industries significantly recognize the effect of AO and OO metrics on software quality. Extensibility enhances the product's functionality. Complexity has an adverse impact on the reliability of application systems. Service Loading behaviour improves product efficiency, Code Weaving affects maintainability, and reusability has a positive impact on the portability of hybrid application systems. Finally, it is recommended that this quality model may be developed and used as an extension of ISO/9126.

7.6 Final Remarks:

The proposed Product Quality Model for hybrid application is not the final work. There are many metrics and features that need to be added to the framework. A clear shortcoming is that hybrid application systems need a quality measurement framework that is compatible with other technical requirements. The weaving process and crosscutting concerns, and all other AO techniques, can be analysed using the proposed product quality model framework. This research is not the final work of identifying AO/OO hybrid application systems; all limitations and barriers will be overcome in future research. Finally, it can be concluded a significant amount of information about AOP and AOP modelling and design in practice. I will carry on with the same energy for the future and go further in this work.

References

[1] ISO/IEC 9126 1, 2001, ISO/IEC 9126 2, 2003, ISO/IEC 9126 3, 2003 and ISO/IEC 9126 4, 2004, "Information Technology – Product Quality Part1: Quality Model, Part 2: External Metrics, Part3: Internal Metrics, Part4: Quality in use Metrics", International Standard ISO/IEC 9126, International Standard Organization.

[2] Aniche, M. (2015, May 1). https://github.com/mauricioaniche/ck. Retrieved April 1, 2019, from https://github.com/mauricioaniche: https://github.com/mauricioaniche/ck

[3] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "Getting started with AspectJ," Communications of the ACM, vol. 44, no. 10, pp. 59–65, 2001.

[4] S. Iqbal, "Aspects and Objects: A Unified Software Design Framework," 2013.

[5] Ghareb, M.I. and Allen, G., 2018, April. State of the art metrics for Aspect oriented programming. In AIP Conference Proceedings (Vol. 1952, No. 1, p. 020107). AIP Publishing.

[6] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G., 2001, June. An overview of AspectJ. In European Conference on Object-Oriented Programming (pp. 327-354). Springer, Berlin, Heidelberg.

[7] Filman, R., Elrad, T., Clarke, S. and Akşit, M., 2004. Aspect-oriented software development. Addison-Wesley Professional.

[8] W. Cazzola, A. Marchetto, and F. B. Kessler, "AOP-HiddenMetrics: Separation, Extensibility, and Adaptability in SW Measurement.," Journal of Object Technology, vol. 7, no. 2, pp. 53–68, 2008.

[9] Walton, P. and Maiden, N. eds., 2019. Integrated software reuse: management and techniques. Routledge.

[10] Ghareb, M.I. and Allen, G., 2018, April. State of the art metrics for Aspect oriented programming. In AIP Conference Proceedings (Vol. 1952, No. 1, p. 020107). AIP Publishing.

[11] Gulia, P., Khari, M. and Patel, S., 2019. Metrics Analysis in Object Oriented and Aspect Oriented Programming. Recent Patents on Engineering, 13(2), pp.117-122.

[12] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," IEEE Transactions on software engineering, vol. 30, no. 8, pp. 491–506, 2004.

[13] M. Ceccato and P. Tonella, "Measuring the effects of software aspectization," in 1st Workshop on Aspect Reverse Engineering, 2004, vol. 12.

[14] A. Mitchell and J. F. Power, "Using object-level runtime metrics to study coupling between objects," in Proceedings of the 2005 ACM symposium on Applied computing, 2005, pp. 1456–1462.

[15] D. Ng, D. R. Kaeli, S. Kojarski, and D. H. Lorenz, "Program comprehension using aspects," in ICSE 2004 Workshop WoDiSEE'2004, 2004.

[16] H. Li, M. Zhou, G. Xu, and L. Si, “Aspect-oriented Programming for MVC Framework,” in Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on, 2010, pp. 1–4.

[17] Rashid, Awais, Ana Moreira, and Jo\=ao Araújo. “Modularisation and Composition of Aspectual Requirements.” In Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, 11–20, 2003.

[18] Jacobson, Ivar. “Use Cases and Aspects-Working Seamlessly Together.” Journal of Object Technology 2, no. 4 (2003): 7–28.

[19] Baniassad, Elisa, and Siobhan Clarke. “Theme: An Approach for Aspect-Oriented Analysis and Design.” In Proceedings of the 26th International Conference on Software Engineering, 158–67, 2004.

[20] M. Berkane, M. Boufaida, and L. Seinturier, “Reasoning about design patterns with an Aspect-Oriented approach,” in Information Technology and e-Services (ICITeS), 2012 International Conference on, 2012, pp. 1–7.

[21] S. A. Khan and A. Nadeem, “UML extensions for modelling of aspect-oriented software: a survey,” in Proceedings of the 2010 National Software Engineering Conference, 2010, p. 5.

[22] M. L. Bernardi and G. A. Di Lucca, “Improving Design Patterns Modularity Using Aspect Orientation,” STEP 2005, p. 209, 2005.

[23] Hannemann, J., Kiczales, G.,’Overcoming the Prevalent Decomposition of Legacy Code’, Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE), 2001.

[24] Hachani, O., Bardou D., ’On Aspect-Oriented Technology and Object-Oriented Design Patterns’, Proc. of European Conference on Object-Oriented Programming (ECOOP), 2003.

[25] Hannemann, J., Kiczales, G., ’Design Patterns Implementation in Java and AspectJ’, Proc. of Object Oriented Programming Systems Languages and Applications (OOPSLA), 2002.

[26] Bernardi, M.L., Cimitile, M., Distance, D., ‘Web applications design recovery and evolution with RE-UWA’, Journal of Software: Evolution and Process, vol. 25, no. 8, pp. 789-814, 2013.

[27] E. W. Dijkstra, A Discipline of Programming. Englewood Cliffs, NJ: Prentice Hall, 1976.

[28] A. Mitchell and J. F. Power, “Using object-level run-time metrics to study coupling between objects,” in Proceedings of the 2005 ACM symposium on Applied computing, 2005, pp. 1456–1462.

[29] Ortu, M., Orrú, M. and Destefanis, G., 2019, February. On Comparing Software Quality Metrics of Traditional vs Blockchain-Oriented Software: An Empirical Study. In 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE) (pp. 32-37). IEEE.

[30] Tsunoda, T., Washizaki, H., Fukazawa, Y., Inoue, S., Hanai, Y. and Kanazawa, M., 2019. Metrics to Predict Future Modifications and Defects Based on Software Requirements Specifications. IEIE Transactions on Smart Processing & Computing, 8(3), pp.210-218.

[31] Coulin, T., Detante, M., Mouchère, W. and Petrillo, F., 2019. Software Architecture Metrics: a literature review. arXiv preprint arXiv:1901.09050.

[32] E. Rønningen and T. Steinmoen, "Increasing readability with Aspect-Oriented Programming," Department of Computer and Information Science (IDI), 2003.

[33] C. Zhang and H.-A. Jacobsen, "Quantifying aspects in middleware platforms," in Proceedings of the 2nd international conference on Aspect-oriented software development, 2003, pp. 130–139.

[34] M. Mickelsson, "Aspect-Oriented Programming compared to Object-Oriented Programming when implementing a distributed, web based application," Department of Information Technology, Uppsala University, 2002.

[35] Y. Coady and G. Kiczales, "Back to the future: a retroactive study of aspect evolution in operating system code," in Proceedings of the 2nd international conference on Aspect-oriented software development, 2003, pp. 50–59.

[36] S. L. Tsang, S. Clarke, and E. Baniassad, "Object metrics for aspect systems: Limiting empirical inference based on modularity," Submitted to ECOOP, 2004.

[37] A. A. Zakaria and H. Hosny, "Metrics for aspect-oriented software design," in Proc. Third International Workshop on Aspect-Oriented Modeling, AOSD, 2003, vol. 3.

[38] R. Burrows, F. C. Ferrari, A. Garcia, and F. Ta"\iani, "An empirical evaluation of coupling metrics on aspect-oriented programs," in Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics, 2010, pp. 53–58.

[39] R. Burrows, A. Garcia, and F. Ta"\iani, "Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies," in International Conference on Evaluation of Novel Approaches to Software Engineering, 2008, pp. 277–290.

[40] K. Dhambri, J.-F. Gélinas, S. Hassaine, and G. Langelier, "Visualization-based Analysis of Quality for Aspect-Oriented Systems."

[41] Nistala, P., Nori, K.V. and Reddy, R., 2019, May. Software quality models: a systematic mapping study. In Proceedings of the International Conference on Software and System Processes (pp. 125-134). IEEE Press.

[42] Fenton, N. and Bieman, J., 2014. Software metrics: a rigorous and practical approach. CRC press.

[43] Edsger W. Dijkstra. A Discipline of Programming. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[44] David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Mod- ules. Comm. ACM, 15(12):1053–1058, December 1972.

[45] Karl J. Lieberherr. Adaptive Object-Oriented Software: the Demeter Method with Propagation Patterns. PWS Publishing Company, Boston, 1996.

[46] Mehmet Ak¸ sit, Lodewijk Bergmans, and Sinan Vural. An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach. In Proc. of the 6th European Conference on Object-Oriented Programming (ECOOP'92), Utrecht, The Netherlands, June/July 1992.

[47] William H. Harrison and Harold L. Ossher. Subject-Oriented Programming - A Critique of Pure Objects. In Proc. of the 8th Conference on Object-Oriented Pro- gramming Systems, Languages, and Applications (OOPSLA'93), Washington, DC, USA, September 1993.

[48] Peri L. Tarr, Harold L. Ossher, William H. Harrison, and Stanley M. Sutton, Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In Proc. of the 21st Int. Conference on Software Engineering (ICSE'99), Los Angeles, California, May 1999.

[49] Kiczales, G., et al. An overview of AspectJ. In Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP). Springer, 2001.

[50] A. Mendhekar, G. Kiczales, and J. Lamping, "RG: A case-study for aspect-oriented programming," 1997.

[51] S. Apel and D. Batory, "How AspectJ is Used: An Analysis of Eleven AspectJ Programs", Technical report MIP-0801, Universität Passau, Passau, Germany, 2008.

[52] M. Kersten, "AOP@Work: AOP tools comparision, Part1," IBM, http://www.ibm.com/developerworks/java/library/j-aopwork1/. [Accessed: Apr. 10, 2015].

[22] A. Oaks and H. Wong, Java Threads, 3rd ed., O'Reilly, 2004.

[53] K. Sirbi and P. J. Kulkarni, "AOP and its impact on software quality," Elixir Computer Science and Engineering, pp. 12606–12610, 2013.

[54] Bhatti, H.R., 2011. Automatic measurement of source code complexity.

[55] Wilhelm, M. and Diehl, S., 2005. Dependency viewer-a tool for visualizing package design quality metrics. In Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on(pp. 1-2). IEEE.

[56] Yadav, V. and Singh, R., 2013, February. Predicting design quality of object-oriented software using UML diagrams. In Advance Computing Conference (IACC), 2013 IEEE 3rd International (pp. 1462-1467). IEEE.

[57] Lincke, R., Lundberg, J., and Löwe, W., 2008, July. Comparing software metrics tools. In Proceedings of the 2008 international symposium on Software testing and analysis (pp. 131-142). ACM.

[58] Kalan, R.S. and Ünalir, M.O., 2016, October. Leveraging big data technology for small and medium-sized enterprises (SMEs). In Computer and Knowledge Engineering (ICCKE), 2016 6th International Conference on (pp. 1-6). IEEE.

[59] Cunningham, H., 2002. GATE, a general architecture for text engineering. Computers and the Humanities, 36(2), pp.223-254.

[60] Ghareb, Mazen and Allen, Gary (2015) Improving the Design and Implementation of Software Systems uses Aspect Oriented Programming. In: Second Scientific Conference University of Human Development, 1st-2nd April/2015, University of Human Development, Sulaymaniyah, Kurdistan Region of Iraq

[61] R. Burrows, A. Garcia, and F. Ta"\iani, “Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies,” in International Conference on Evaluation of Novel Approaches to Software Engineering, 2008, pp. 277–290.

[62] Eaddy, M., Aho, A., and Murphy, G.C., 2007, May. Identifying, assigning, and quantifying crosscutting concerns. In Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques (p. 2). IEEE Computer Society.

[63] Chidamber, S.R., and Kemerer, C.F., 1994. A metrics suite for object-oriented design. IEEE Transactions on software engineering, 20(6), pp.476-493.

[64] Ghareb, M. and Allen, G., 2015. Improving the Design and Implementation of Software Systems uses Aspect Oriented Programming.

[65] A. Kaur, S. Singh, K. Kahlon, and P. S. Sandhu, “Empirical Analysis of CK \& MOOD Metric Suit,” International Journal of Innovation, Management and Technology, vol. 1, no. 5, p. 447, 2010.

[66] Chidamber, S.R., and Kemerer, C.F., 1991. Towards a metrics suite for object-oriented design (Vol. 26, No. 11, pp. 197-211). ACM.

[67] Subramanyam, R. and Krishnan, M.S., 2003. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on software engineering, 29(4), pp.297-310.

[68] Zhao, J., 2004, September. Measuring coupling in aspect-oriented systems. In 10th International Software Metrics Symposium (Metrics 04).

[69] Rothenberger, M.A., Dooley, K.J., Kulkarni, U.R. and Nada, N., 2003. Strategies for software reuse: A principal component analysis of reuse practices. IEEE Transactions on Software Engineering, 29(9), pp.825-837.

[70] Mens, T. and Tourwé, T., 2004. A survey of software refactoring. IEEE Transactions on software engineering, 30(2), pp.126-139.

[71] S. MacDonell,M. Shepperd, P. Sallis, Metrics for database systems an empirical study, in: Proc. 4th Int'l Software Metrics Symposium (Metrics 97), IEEE Computer Society Press, Albuquerque, May 1997,pages 99–107.

[72] L. Briand, S. Morasca, V. Basili, An operational process for goaldriven definition of measures, IEEE Transactions on Software Engineering 28 (12) (2002) 1106–1125.

[73] D. de Champeaux, Object-Oriented Development Process and Metrics, Prentice-Hall, Upper Saddle River, 1997.

[74] S. Pfleeger, Guest editor's introduction: assessing measurement, IEEE Software 14 (2) (1997) 25–26.

[75] ISO/IEC, International Standard ISO/IEC 90003, Software and Systems Engineering—Guidelines for the Application of ISO/IEC 9001:2000 to Computer Software, International Standards Organization, Geneva, Switzerland, 2004.

[76] Li, W., 1999. Software product metrics. IEEE Potentials, 18(5), pp.24-27.

[77] Honglei, T., Wei, S. and Yanan, Z., 2009, December. The research on software metrics and software complexity metrics. In 2009 International Forum on Computer Science-Technology and Applications (Vol. 1, pp. 131-136). IEEE.

[78] Fenton, N. and Bieman, J., 2014. Software metrics: a rigorous and practical approach. CRC press.

[79] Côté, Vianney, Pierre Bourque, Serge Oligny, and N. Rivard. "Software metrics: an overview of recent results." Journal of Systems and Software 8, no. 2 (1988): 121-131.

[80] Li, H.F. and Cheung, W.K., 1987. An empirical study of software metrics. IEEE Transactions on Software Engineering, (6), pp.697-708.

[81] Ince, D., 1990. Software metrics: introduction. Information and Software Technology, 32(4), pp.297-303.

[82] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. JSS, 83(1):2–17, 2010.

[83] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. T. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In T. Gyim´othy and A. Zeller, editors, SIGSOFT FSE, pages 4–14. ACM, 2011.

[84] A. E. Hassan. Predicting faults using the complexity of code changes. In ICSE, pages 78–88. IEEE, 2009.

[85] A. Mockus and D. M. Weiss. Predicting risk of software changes. Bell Labs Technical Journal, 5(2):169–180, 2000.

[86] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In ASE'2011,pages 362–371. IEEE, 2011.

[87] Harrison, W., Magel, K., Kluczny, R. and DeKock, A., 1982. Applying software complexity metrics to program maintenance. Computer, (9), pp.65-79.

[88] Boehm, B.W., Brown, J.R. and Lipow, M., 1976, October. Quantitative evaluation of software quality. In Proceedings of the 2nd international conference on Software engineering (pp. 592-605). IEEE Computer Society Press.

[89] Kan, S.H., 2002. Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc..

[90] Neelamegam, C. and Punithavalli, M., 2009. A survey-object oriented quality metrics. Global Journal of Computer Science and Technology, 9(4), pp.183-186.

[91] Shaheen, A., Qamar, U., Nazir, A., Bibi, R., Ansar, M. and Zafar, I., 2019, May. OOCQM: Object Oriented Code Quality Meter. In International Conference on Computational Science/Intelligence & Applied Informatics (pp. 149-163). Springer, Cham.

[92] Ceccato, M. and Tonella, P., 2004, November. Measuring the effects of software aspectization. In 1st Workshop on Aspect Reverse Engineering (Vol. 12).

[93] Shen, H. and Zhao, J., 2007. An evaluation of coupling metrics for aspect-oriented software. Center for Soft. Eng., SJTU, Shanghai-

[94] China, Tech. Rep. SJTU-CSE-TR-07-04.Bartsch, M. and Harrison, R., 2006, March. An evaluation of coupling measures for AspectJ. In LATE Workshop AOSD.

[95] Babu, C. and Vijayalakshmi, R., 2008. Metrics-based design selection tool for Aspect oriented software development. ACM SIGSOFT Software Engineering Notes, 33(5), p.4.

[96] Bartsch, M. and Harrison, R., 2006, March. An evaluation of coupling measures for AspectJ. In LATE Workshop AOSD.

[97] Sirbi, K.S. and Kulkarni, P.J., 2010. Coupling Metrics for Aspect Oriented Programming. International Journal of Advanced Research in Computer Science, 1(3).

[98] Burrows, R., Ferrari, F.C., Garcia, A. and Taïani, F., 2010, May. An empirical evaluation of coupling metrics on aspect-oriented programs. In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (pp. 53-58).

[99] Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C. and von Staa, A., 2006. Modularizing design patterns with aspects: a quantitative study. In Transactions on Aspect-Oriented Software Development I (pp. 36-74). Springer, Berlin, Heidelberg.

[100] Piveta, E.K., Moreira, A., Pimenta, M.S., Araújo, J., Guerreiro, P. and Price, R.T., 2012. An empirical study of aspect-oriented metrics. Science of Computer Programming, 78(1), pp.117-144.

[101] Boticki, I., Katic, M. and Martin, S., 2012. Exploring the educational benefits of introducing aspect-oriented programming into a programming course. IEEE transactions on education, 56(2), pp.217-226.

[102] Mguni, K. and Ayalew, Y., 2013. An assessment of maintainability of an aspect-oriented system. ISRN Software Engineering, 2013.

[103] Gaia, F.N., Ferreira, G.C.S., Figueiredo, E. and de Almeida Maia, M., 2014. A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. Science of Computer Programming, 96, pp.230-253.

[104] Sant'Anna, C., Figueiredo, E., Garcia, A. and Lucena, C.J., 2007, September. On the modularity of software architectures: A concern-driven measurement framework. In European Conference on Software Architecture (pp. 207-224). Springer, Berlin, Heidelberg.

[105] Teebiga, R. and Velan, S.S., 2016, May. Comparison of applying design patterns for functional and non-functional design elements in Java and AspectJ programs. In 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT) (pp. 751-757). IEEE.

[106] Nuñez-Varela, A.S., Pérez-Gonzalez, H.G., Martínez-Perez, F.E. and Soubervielle-Montalvo, C., 2017. Source code metrics: A systematic mapping study. Journal of Systems and Software, 128, pp.164-197.

[107] Kaur, P.J., Kaushal, S., Sangaiah, A.K. and Piccialli, F., 2018. A framework for assessing reusability using package cohesion measure in Aspect oriented systems. International Journal of Parallel Programming, 46(3), pp.543-564.

[108] Da Silva, B.C., Sant'Anna, C. and Chavez, C., 2011, May. Concern-based Cohesion as change proneness indicator: an initial empirical study. In Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics (pp. 52-58).

[109] Sheshasaayee, A. and Jose, R., 2015. A Descriptive Study about Aspect Oriented Coupling and Cohesion Measures. International Journal of Innovative Science, Engineering & Technology, 2(8).

[110] Kumar, A., Kumar, R. and Grover, P., 2008. A critical review of cohesion measures and measurement frameworks in aspect-oriented systems. In Proc. 2nd National Conference on Computing for National Development (INDIACom-2008) (pp. 415-420).

[111] Sant'Anna, C., Figueiredo, E., Garcia, A. and Lucena, C.J., 2007, September. On the modularity of software architectures: A concern-driven measurement framework. In European Conference on Software Architecture (pp. 207-224). Springer, Berlin, Heidelberg.

[112] Sant'Anna, C., Garcia, A. and Lucena, C. 2008. Evaluating the Efficacy of Concern-Driven Metrics: A Comparative Study. In Proc. of the 2nd Workshop on Contemporary Modularization Techniques (ACOM'08), Nashville, USA.

[113] Robillard, M.P. and Murphy, G.C., 2007. Representing concerns in source code. ACM Transactions on Software Engineering and Methodology (TOSEM), 16(1), pp.3-es.

[114] McCabe, T.J., 1976. A complexity measure. IEEE Transactions on software Engineering, (4), pp.308-320.

[115] Harrison, W.A. and Magel, K.I., 1981. A complexity measure based on nesting level. ACM Sigplan Notices, 16(3), pp.63-74.

[116] Howatt, J.W. and Baker, A.L., 1989. Rigorous definition and analysis of program complexity measures: an example using nesting. Journal of Systems and Software, 10(2), pp.139-150.

[117] Piwowarski, P., 1982. A nesting level complexity measure. ACM Sigplan Notices, 17(9), pp.44-50.

[118] Lincke, R., Lundberg, J. and Löwe, W., 2008, July. Comparing software metrics tools. In Proceedings of the 2008 international symposium on Software testing and analysis (pp. 131-142). ACM.

[119] Fenton, Norman, and James Bieman. Software metrics: a rigorous and practical approach. CRC Press, 2014.

[120] Yeresime, Suresh, Jayadeep Pati, and Santanu Ku Rath. "Review of software quality metrics for object-oriented methodology." In Proceedings of International Conference on Internet Computing and Information Communications, pp. 267-278. Springer India, 2014.

[121] Braude, Eric J., and Michael E. Bernstein. Software engineering: modern approaches. Waveland Press, 2016.

[122] Lee, Ming-Chang. "Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance." British Journal of Applied Science & Technology 4, no. 21 (2014): 3069-3095.

[123] E. Rønningen and T. Steinmoen, “Increasing readability with Aspect-Oriented Programming,” Department of Computer and Information Science (IDI), 2003.

[124] S. L. Tsang, S. Clarke, and E. Baniassad, “Object metrics for aspect systems: Limiting empirical inference based on modularity,” Submitted to ECOOP, 2004.

[125] R. Burrows, A. Garcia, and F. Ta"\iani, “Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies,” in International Conference on Evaluation of Novel Approaches to Software Engineering, 2008, pp. 277–290.

[126] Ghareb, M. and Allen, G., 2019. An empirical evaluation of metrics on aspect-oriented programs. UHD Journal of Science and Technology, 3(2), pp.74-86.

[127] K. Lieberher, D. Orleans, and J. Ovlinger, “Aspect Oriented Programming with Adaptive Methods,” Com munications of the ACM, Vol.44, No.10, pp.39 41, Octo ber 2001.

[128] Alexander, R.T., “The Real Costs of Aspect Oriented Programming”, IEEE Software, Vol.20, Issue.6, pp. 91–93, 2003.

[129] Avadhesh Kumar, Rajesh Kumar, and P.S. Grover, “To wards a Unified Framework for Complexity Measurement in Aspect Oriented Systems” , 2008 International Confe rence on Computer Science & Software Engineering (CSSE 2008), Wuhan, China, Dec 12 14, 2008, pp.98 103, IEEE Computer Society.

[130] McCall, J. A., Richards, P. K., and Walters, G. F., “Factors in Software Quality”, Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977.

[131] Boehm, B. W., Brown, J. R., and Lipow, M. L., “Quantit ative Evaluation of Software Quality”, Proceedings of the 2nd International Conference on Software Engineer ing, San Francisco, California, United States, 1976, pp. 592 605, IEEE Computer Society Press.

[132] Dromey, R. G., “A Model for Software Product Qual ity”, IEEE Transactions on Software Engineering, Vol. 21, No.2, pp.146 162, Feb.1995.

[133] IEEE, Guide to the Software Engineering Body of Knowledge, 2001, www.swebok.org.

[134] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G., “An Overview of AspectJ”, In Proc. of the 15th European Conference on Object Oriented Programming, Lecture Notes in Computer Science, Vol. 2072. Springer Verlag, London, pp. 327 353, June 18 22, 2001.

[135] Deutsch, M. S., and Wills, R. R., “Software Quality Engineering; A Total Technical and Management Ap-proach”, Prentic-Hall, Inc. 1998.

[136] Word, W. A., and Venkataraman, B., “Some Observa-tions on Software Quality”, in Proc. of the 37th Annual Southeast regional conference, April 1999.

[137] Arun Sharma, Rajesh Kumar, and P. S. Grover, “Esti-mation of Quality for Software Components: an Empiri-cal Approach”, ACM SIGSOFT Software Engineering Notes, Vol. 33, Issue.6, pp.1-10, 2008.

[138] Chang, C., Wu, C., and Lin, H., “Integrating Fuzzy Theory and Hierarchy Concepts to Evaluate Software Quality”, Software Quality Control, Vol. 16, Issue.2, pp.263-276, Jun. 2008.

[139] Adnan Rawashdeh, Bassem Matalkah, “A New Soft-ware Quality Model for Evaluating COTS Components”, Journal of Computer Science, Vol. 2 Issue. 4, pp.373-381, 2006.

[140] M. Bertoa, A. Vallecillo, “Quality Attributes for COTS Components”, in the Proceedings of the 6th In-ternational ECOOP Workshop on Quantitative Ap-proaches in Object-Oriented Software Engineering (QAOOSE), Spain, 2002.

[141] Adnan Rawashdeh, Bassem Matalkah, “A New Soft-ware Quality Model for Evaluating COTS Components”, Journal of Computer Science, Vol. 2 Issue. 4, pp.373-381, 2006.

[142] M. Bertoa, A. Vallecillo, “Quality Attributes for COTS Components”, in the Proceedings of the 6th In-ternational ECOOP Workshop on Quantitative Ap-proaches in Object-Oriented Software Engineering (QAOOSE), Spain, 2002.

[143] Kumar, A., Grover, P.S. and Kumar, R., 2009. A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO). ACM SIGSOFT Software Engineering Notes, 34(5), pp.1-9.

[144] P.S. Grover, Rajesh Kumar, and Avadhesh Kumar, “Measuring Changeability for Generic Aspect-Oriented Systems”, ACM SIGSOFT Software Engineering Notes, Volume 33, Issue 6, pp-1-5, November 2008.

[145] Ghareb, M.I. and Allen, G., 2021. Quality Metrics measurement for Hybrid Systems (Aspect Oriented Programming–Object Oriented Programming). Technium: Romanian Journal of Applied sciences and Technology, 3(3), pp.82-99

[146] Basili, V.R., Selby, R.W. and Hutchens, D.H., 1986. Experimentation in software engineering. IEEE Transactions on software engineering, (7), pp.733-743.

[147] Wohlin C, Höst M, Ohlsson MC, Regnell B, Runeson P, Wesslén A (2000) Experimentation in software engineering—an introduction. Kluwer

[148] Tichy WF (1998) Should computer scientists experiment more? Computer 31(5):32–40 doi:10.1109/2.675631

[149] Seaman C (1999) Qualitative methods in empirical studies of software engineering. IEEE Trans Softw Eng 25(4):557–572 see also Chapter 2 in Shull et al.

[150] Lethbridge, T.C., Sim, S.E. and Singer, J., 2005. Studying software engineers: Data collection techniques for software field studies. Empirical software engineering, 10(3), pp.311-341.

[151] Easterbrook, S., Singer, J., Storey, M.A. and Damian, D., 2008. Selecting empirical methods for software engineering research. In Guide to advanced empirical software engineering (pp. 285-311). Springer, London.

[152] Zelkowitz, M.V. and Wallace, D.R., 1998. Experimental models for validating technology. Computer, 31(5), pp.23-31.

[153] Flyvbjerg, B., 2007. Five misunderstandings about case-study research In Qualitative Research Practice, concise paperback edition.

[154] Host, M. and Runeson, P., 2007, September. Checklists for software engineering case study research. In First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007) (pp. 479-481). IEEE.

[155] Kumar, A., Kumar, R. and Grover, P.S., 2008, December. Towards a unified framework for complexity measurement in aspect-oriented systems. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering-Volume 02 (pp. 98-103).

[156] Shaheen, A., Qamar, U., Nazir, A., Bibi, R., Ansar, M. and Zafar, I., 2019, May. OOCQM: Object Oriented Code Quality Meter. In International Conference on Computational Science/Intelligence & Applied Informatics (pp. 149-163). Springer, Cham.

[157] Burrows, R., Ferrari, F.C., Garcia, A. and Taïani, F., 2010, May. An empirical evaluation of coupling metrics on aspect-oriented programs. In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (pp. 53-58). ACM.

[158] Canfora, G. and Cerulo, L., 2005, September. How crosscutting concerns evolve in JHotDraw. In 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05) (pp. 65-73). IEEE.
[159] Kumar, P. and Singh, S.K., 2016, April. A systematic assessment of aspect-oriented software development (AOSD) using JHotDraw application. In 2016 International Conference on Computing, Communication and Automation (ICCCA) (pp. 779-784). IEEE.
[160] Gana, K. and Broc, G., 2019. Structural equation modeling with lavaan. John Wiley & Sons.
[161] Jnr, B.A., 2019. Validating the usability attributes of AHP-software risk prioritization model using partial least square-structural equation modeling. Journal of Science and Technology Policy Management.
[162] Rosseel, Y., 2020. Small sample solutions for structural equation modeling. SMALL SAMPLE SIZE SOLUTIONS, p.226.
[163] Collier, J.E., 2020. Applied Structural Equation Modeling using AMOS: Basic to Advanced Techniques. Routledge.
[164] Javed, M., 2014. Towards the maturity model for feature oriented domain analysis. Computational Ecology and Software, 4(3), p.170.
[165] D. G. Altman and J. M. Bland, “Standard deviations and standard errors,” Bmj, vol. 331, no. 7521, p. 903, 2005.
[166] Crede, M. and Harms, P., 2019. Questionable research practices when using confirmatory factor analysis. Journal of Managerial Psychology.
[167] Arumi, E.R., Sukmasetya, P. and Setiawan, A., 2020. Model Evaluasi Usability Menggunakan Confirmatory Factor Analysis pada KRS Online. Jurnal Teknologi Informasi dan Ilmu Komputer, 8(1).
[168] Mohajeri, K., Mesgari, M. and Lee, A.S., 2020. When Statistical Significance Is Not Enough: Investigating Relevance, Practical Significance, and Statistical Significance. MIS Quarterly, 44(2).
[169] Evans, S., Johnson, A.L., Anderson, J.M., Checketts, J.X., Scott, J., Middlemist, K., Fishbeck, K. and Vassar, M., 2020. The Potential Effect of Lowering the Threshold of Statistical Significance from P< 0.05 to P< 0.005 in Orthopaedic Sports Medicine. Arthroscopy: The Journal of Arthroscopic & Related Surgery.
[170] Ahlgren, P., Jarneving, B. and Rousseau, R., 2003. Requirements for a cocitation similarity measure, with special reference to Pearson's correlation coefficient. Journal of the American Society for Information Science and Technology, 54(6), pp.550-560.
[171] Sedgwick, P., 2012. Pearson’s correlation coefficient. Bmj, 345, p.e4483.
[172] Valentini, Felipe, and Bruno Figueiredo Damasio. "Average Variance Extracted and Composite Reliability: Reliability Coefficients/Variancia Media Extraida e Confiabilidade Composta: Indicadores de Precisao." Psicologia: Teoria e Pesquisa 32, no. 2 (2016).

[173] Hair Jr, J.F., Howard, M.C. and Nitzl, C., 2020. Assessing measurement model quality in PLS-SEM using confirmatory composite analysis. Journal of Business Research, 109, pp.101-110.
[174] Lee, S.L., Lin, S.Y., Ko, H.K. and Liu, Y.Y., 2020. Construct validity and reliability of the Chinese version Personal Adjustment and Role Skills Scale III for adolescents with chronic disease. Journal of Pediatric Nursing.
[175] Lai, M.H., 2020. Composite reliability of multilevel data: It's about observed scores and construct meanings. Psychological Methods.
[176] Fornell, C. and Larcker, D.F., 1981. Evaluating structural equation models with unobservable variables and measurement error. Journal of marketing research, 18(1), pp.39-50.
[177] Netemeyer, R.G., Bearden, W.O. and Sharma, S., 2003. Scaling procedures: Issues and applications. Sage Publications.
[178] Cronholm, S. and Goldkuhl, G., 2003. Strategies for information systems evaluation-six generic types. Electronic Journal of Information Systems Evaluation, 6(2), pp.65-74.
[179] Chen, S., Osman, N.M., Nunes, J.M.B. and Peng, G.C., 2011. Information systems evaluation methodologies. In Proceedings of the IADIS International Workshop on Information Systems Research Trends, Approaches and Methodologies. Sheffield.
[180] Harrison, W.A. and Magel, K.I., 1981. A complexity measure based on nesting level. ACM Sigplan Notices, 16(3), pp.63-74.
[181] Howatt, J.W. and Baker, A.L., 1989. Rigorous definition and analysis of program complexity measures: an example using nesting. Journal of Systems and Software, 10(2), pp.139-150.