

# **SYSTEMATIC METRICS ANALYSIS OF OBJECT AND ASPECT ORIENTED PARADIGM**

*A Thesis Submitted*

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE  
OF  
**DOCTOR OF PHILOSOPHY**

IN

**COMPUTER SCIENCE AND ENGINEERING**

**SUBMITTED BY:**

SHRIKANT PATEL

18SCSE3010028

(Enrollment No-18053010073)

**UNDER THE SUPERVISION OF:**

PROF. (DR.) KAVITA

(SUPERVISOR)

PROF. (DR.) SANJAY KUMAR

(CO-SUPERVISOR)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
[SCHOOL OF COMPUTING SCIENCE & ENGINEERING]**

**GALGOTIAS UNIVERSITY**

**PLOT NO. 2, SECTOR 17-A, YAMUNA EXPRESWAY**

**GREATER NOIDA, UTTAR-PRADESH**

**INDIA**

**JULY, 2023**



## **CERTIFICATE**

This is to certify that **SHRIKANT PATEL (Reg.No.-18SCSE3010028)** has presented his pre-submission seminar of the thesis entitled “**SYSTEMATIC METRICS ANALYSIS OF OBJECT AND ASPECT ORIENTED PARADIGM**” before the committee and summary is approved and forwarded to School Research Committee of Computing Science & Engineering, in the Faculty of Engineering & Technology, Galgotias University-Uttar Pradesh.

Dean – SCSE

Dean – Ph.D. & PG

The Ph.D. Viva-Voice examination of **SHRIKANT PATEL**, Research Scholar has been held on\_\_\_\_\_.

Supervisor

External Examiner

Co-Supervisor



## **CANDIDATE DECLARATION**

I hereby declare that the work which is being presented in the thesis, entitled “**SYSTEMATIC METRICS ANALYSIS OF OBJECT AND ASPECT ORIENTED PARADIGM**” in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** in Computer Application and submitted in Galgotias University, Uttar Pradesh is an authentic record of my own work carried out from APRIL 2019 under the supervision of **Dr. Kavita**, Professor, School of Computing Science & Engineering, Galgotias University and co-suprvision of **Dr. Sanjay Kumar**, Professor-GNIT. The matter embodied in this thesis has not been submitted by me for the award of any other degree or from any other University.

**SHRIKANT PATEL**  
**18SCSE3010028**

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

**DR. KAVITA**  
Supervisor  
SCSE  
Galgotias University

**DR. SANJAY KUMAR**  
Co-Supervisor  
GNIT



## **APPROVAL SHEET**

This Thesis entitled **SYSTEMATIC METRICS ANALYSIS OF OBJECT AND ASPECT ORIENTED PARADIGM** by **SHRIKANT PATEL** is approved for the Degree of Doctor of Philosophy.

**Examiner**

**Supervisor**

**Co-Supervisor**

**Chairman**

## ACKNOWLEDGEMENTS

Doing work as a Research Scholar for the degree of Ph.D. in Galgotias University was quite magnificent and challenging experience for me. In all these years, many people directly or indirectly contributed in shapping up my research work. It was hardly possible for me to complete my doctoral work without the precious and invaluable support of these personalities.

I would like to express my gratitude to my guide **Dr. Kavita** and co-guide **Dr. Sanjay Kumar** for their guidance, kindness, motivations, suggestions and insight throughout this Ph.D. research, as without them this thesis would not have been completed. I offer my sincere thanks to my family members for their inspiration and moral support. Thank you all for your strong support.

I must owe a special debt of gratitude to Hon'ble Chancellor **Mr. Suneel Galgotia**, CEO **Mr. Dhruv Galgotia**, Hon'ble Vice-Chancellor **Dr. K. Mallikharjuna Babu**, Galgotias University for their valuable support throughout my research work.

I express my sincere thanks to **Dr. Munish Sabharwal**, Dean School of Computing Science & Engineering and **Dr. Alok Katiyar**, Coordinator-Ph.D. for their guidance and moral support during my research work and all faculties of School of Computing Science & Engineering who helped me a lot in my course of research work and all those who stood behind me.

Nothing is possible without the constant support of my family. I would like to convey my deep regard to my parents for their wise counsel and indispensable advice that always encouraged me to work hard for the completion of my research work. My highest gratitude goes to my parents and all my family members for their relentless support, blessings and encouragement. Special mention goes to my mother **Smt. Vidyawati Patel**, my wife **Rajni Gour Patel** and my kids **Ojasvi and Kushagra**. My special thanks to all my friends, and to all those who stood behind by me like a support and helped me in completing this research work.

**SHRIKANT PATEL**

## ABSTRACT

Measuring characteristics that are essential to a software project's success is made possible by software metrics. The characteristics and connections between them become clearer when these attributes are measured. This, in turn, helps people make better decisions. Inconsistent, irregular, and infrequent measurements have an impact on software engineering. Software testing becomes the necessary portion of software development because it lets process attributes be measured. The management can gain a deeper understanding of the software testing procedure by measuring its attributes. This research aims to develop, execute and validate metrics as well as methodologies for recognizing few significant aspects that influence software development, estimating the effect of user-initiated variations on a software system. The work presented in the thesis strategies to help with decisions that affect software improvement.

Further, research addresses the following matters: Analyzing the scope to which shifting necessities influence the design of a model, guiding the delegation of responsibilities to software components, combining Aspect Oriented Programming (AOP) and Object Oriented Programming (OOP) to best offer a model's operation, as well as determining whether and how outsourced and offshore development influences a system's design. This work on metrics and methods serve as heuristics throughout the life cycle phase of software improvement, assisting experts in selecting possibilities and making decisions. Test development pprocedures as well as software test planning on employee management system have a variety of measurable attributes, according to the survey of the literature. For the purposes of the software test planning and test design processes, the study divided these characteristics into multiple categories. Currently available measurements are examined for each of these characteristics.

This thesis presents a consolidation of these measurements with the intention of providing management with an opportunity to consider process enhancements. With increased use of software applications, the quality assessment of software, such as gain consequence, defect measurement. In numerous empirical studies of software products, measurement of metrics are regarded as the main pointer of software maintenance as well as imperfection prediction. AOP becomes one of the novel development approaches, but no one can agree that which metrics considered to be as reliable quality indicators. By providing a newly developed system constructs like inter-type, advice and point cuts associations, AOP hopes to improve programming quality. As a result, it is unclear whether direct expansions of conventional OO

evaluations can yield quality pointers for AOP. However, established coupling measurements are frequently used in AOP research.

In spite of the fact that AOP has only recently been used in pragmatic research, coupling cohesion have been used as beneficial indicators of fault inclination in this perspective. This research examinethe most recent metrics for measuring the development of Aspect Oriented systems.

*Keywords:* Aspect-Oriented Programming, Object Oriented Programming, Line of Code, Employee Management System, AsectJ, Javascript, Cohesion, Coupling.

## LIST OF FIGURES

<b>FIGURE NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
2.1	Representation of Aspect Weaver	15
2.2	Weaving in AOP	17
2.3	Dimensions of Separation of Concerns	44
2.4	Aspect Weaving	45
3.1	Project Structure of Employee Management System	47
3.2	Application Class of Employee Management System	48
3.3	Bean Class of Employee Management System	49
3.4	Working of Controller Class in an Employee Management System	55
3.5	Controller class of Employee Management System	56
3.6	Aspect Class of Employee Management System	59
3.7	Service Class of Employee Management System	60
3.8	Architectural Diagram of Employee Management System	60
3.9	Project Flow Diagram of Employee Management System	61
3.10	Work Flow of an Employee Management System	62
3.11	Required Aspect Interface	63
3.12	Application Class of EMS	63
3.13	Bean Class of EMS	64
3.14	Working of Controller in an Employee Managent System	65
3.15	Controller Class	65
3.16	Aspect Class	66
3.17	Service Class	67
3.18	Archtectural Design of EMS	68
3.19	Work Flow Diagram of EMS	71
3.20	Work Flow Daigram of EMS	73
3.21	Required Aspect Interface	77
4.1	Difficulty Level between AOP and OOP	83
4.2	Comparison of Effort Level in between AOP and OOP	84



4.3	Comparison of Length of Program between AOP and OOP	85
4.4	Comparison of Line of Code between and AOP and OOP	86
4.5	Comparison of Program Level between and AOP and OOP	87
4.6	Comparison of Size of Vocabulary between and AOP and OOP	87
4.7	Comparison of Time Taken between and AOP and OOP	88
4.8	Comparison of Volume between and AOP and OOP	90

## LIST OF TABLES

<b>TABLE NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
1.1	AOP Software Quality Metrics	3
2.1	Existing Studies towards Software Metrics	21
2.2	AO Abstractions and Mechanisms Unique to Three Main AOP Languages	25
2.3	Categories of Software Maintenance (IEEE Std. 1219-1998)	30
2.4	Categories of Software Maintenance	30
2.5	Summary of Maintainability Metrics	32
4.1	The Metrics Suite	79

## LIST OF ABBREVIATIONS

AOP	Aspect Oriented Programming
OOP	Object Oriented Programming
AO	Aspect-Oriented
OO	Object-Oriented
CFG	Control Flow Graph
DFG	Dependency Flow Graph
DG	Dependence Graph
SOC	Separation Of Concerns
AOSD	Aspect Oriented Software Design
UML	Unified Modelling Language
CBO	Coupling Between Objects
WMC	Weighted Methods Class
LOC	Lines of Code
POF	Polymorphism Factor
COF	Coupling Factor
MIF	Method Inheritance Factor
MHF	Method Hiding Factor
AIF	Attribute Inheritance Factor
AHF	Attribute Hiding Factor
DSC	Design Size of Class
NOH	Number of Hierarchies
MOP	Number of Polymorphic
NOM	Number of Methods
MFA	Measure of Functional Abstraction
DCC	Direct Class Coupling
CIS	Class Interface Size

MOA	Measure of Aggregation
DAM	Data Access Metric
CAM	Cohesion Among Methods
ANA	Average Number of Ancestors
DIT	Depth of Inheritance Tree
NOM	Number of Methods
NOA	Number of Attributes
CC	Class Complexity
WMC	Weight Method Complexity
LCOM	Lack of Method Cohesion
LCC	Loss Class Cohesions
TCC	Tight Class Cohesion
NMO	Number of Methods Overridden
AHF	Attribute Hiding Factor
MHF	Method Hiding Factor
CF	Coupling Factor
MPC	Message Passing Coupling
RFC	Response For the Class
DAC	Data Abstract Coupling
CBO	Coupling Between Objects
IDE	Integrated Development Environments
IACFG	Inter-procedural Aspect Control Flow Graph
SMI	Software Maturity Index
GQM	Goal-Question-Metric method
TSP	Team Software Process
PSP	Personal Software Process
FEAST	Feedback, Evolution, and Software Technology

PSI	Probability Of Success Indicator
HR	Human Resource
SRS	Software Requirements Specifications
ERD	Entity Relationship Diagram
DFD	Data Flow Diagram
PHP	Hypertext Preprocessor
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
PHP	Hypertext Preprocessor
CSS	Cascading Style Sheets
RAD	Rapid Action Development
VS	Vocabulary Size
OP	Number Of Class Operations
NOM	Number Of Modules
NCLOC	Non-Commented Lines Of Code
LM	Lines Of Code Modified
LR	Lines Of Code Removed
LA	Lines Of Code Added
OM	Operations Modified
OR	Operations Removed
OA	Operations Added
MA	Modules Added
MM	Modules Modified
MR	Modules Removed
NOA	Number Of Attributes

# TABLE OF CONTENTS

<b>Certificate</b>	<b>ii</b>
<b>Candidate Declaration</b>	<b>iii</b>
<b>Approval Sheet</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Abbreviations</b>	<b>xiv</b>

<b>CHAPTER 1- INTRODUCTION .....</b>	<b>1</b>
1.1 History .....	2
1.2 Problem Statement.....	6
1.3 Aims Of This Research.....	8
1.4 Main Contributions .....	8
1.5 Research Outline.....	9
1.6 AOP Quality Metrics .....	11
1.7 Outline of the Thesis.....	12

<b>CHAPTER 2-LITERATURE REVIEW .....</b>	<b>14</b>
2.1 Aspect Oriented Programming .....	15
2.1.1 Relevant Concepts .....	15
2.1.2 Development Process with Aspects.....	16
2.2 The AO Paradigm and Software Development .....	16
2.2.1 AOP and Software Development Process .....	17
2.2.2 Development.....	17
2.2.3 Testing .....	17
2.2.4 AOP and Unified Modelling Language .....	18
2.3 Object Oriented Programming.....	18
2.4 Design Patterns .....	20
2.5 Metrics .....	21
2.6 AOP Languages and Constructs .....	24
2.7 Fundamental Concepts in AOP .....	26
2.8 Maintenance of Aspect-Oriented Programs.....	28
2.9 Current research work on AOP.....	30
2.9.1 Maintainability Metrics .....	30

2.9.2 Empirical Studies.....	31
2.10 Deeper Perception on Software Metric Development .....	33
2.11 Metrics in the New Millennium.....	39
2.12 Crosscutting Concerns .....	42
<b>CHAPTER 3 -METHODOLOGY TO DESIGN EMS.....</b>	<b>46</b>
3.1 Planning.....	48
3.2 Methodology.....	50
3.2.1 Spring AOP capabilities and goals .....	55
3.3 Thesis Structure .....	58
3.4 Classes .....	61
3.4.1Application Class.....	61
3.4.2 Bean Class .....	64
3.4.3 Controller Class .....	64
3.4.4 Aspect Class: .....	66
3.4.5 Service Class:.....	66
3.5 Architecture Diagram .....	68
3.6 Project Flow Diagram.....	70
3.7 Data Flow Diagram (DFD).....	73
3.8 Aspect Interface.....	76
<b>CHAPTER 4-RESULTS &amp; DISCUSSIONS .....</b>	<b>78</b>
4.1 Mathematical Expressions used.....	78
4.2 Results .....	81
4.2.1 Difficulty .....	82
4.2.2 Effort.....	83
4.2.3Length of Program.....	84
4.2.3 Lines of Codes .....	85
4.2.4 Program Level .....	86
4.2.5 Size of Vocabulary .....	87
4.2.6 Time Taken.....	88
4.2.7 Volume .....	89
4.3 Assessment of Metrics.....	90
4.4 Threats to Validity .....	91

<b>CHAPTER 5-CONCLUSIONS &amp;FUTURE DIRECTIONS .....</b>	<b>93</b>
5.1 DISCUSSION .....	93
5.2 CONCLUSION.....	94
5.3 FUTURE DIRECTIONS .....	95
<b>REFERENCES .....</b>	<b>97</b>
<b>LIST OF PUBLICATIONS.....</b>	<b>107</b>



# CHAPTER 1

## INTRODUCTION

Software maintenance becomes the procedure of performing ugradations to a software product afterwards it has been provided to repair bugs, increase the efficiency or other features, or acclimatize the model into a new settings. The fundamental understanding of software maintenance is that it encompasses more than just the program. It's easy to think of activities that are only done on programs, like writing object code, source code, and reports for any aspect of the user manuals, design, specification and requirements assessment. Re-engineering techniques like re-use, refactoring, slicing, and re-engineering are probably used in software maintenance. Software system maintainability is known for being disreputablyproblematic to define. It may be simple to compute some aspects. However, there are numerous techniques to measure practically, especially for complicated software. The maintenance activity will also be measured in a different way because of the different development method. For instance, procedural method development differs from object-oriented measurement development. Even so, object-oriented is distinct from aspect-oriented.

Kiczales et al. proposed the aspect-oriented (AO) paradigm after the object-oriented (OO) paradigm with the intention of improving software maintainability by incorporating cross-cutting concern into new modularization mechanisms [1]. Leading the reduction of complications throughout the development cycle, particularly during the maintenance stage, is extremely beneficial in software engineering. Separation of concerns is capable of detecting, capturing, and controlling only those software components that are pertinent to a given perception, goal, or intention. Its goal is to end the dominant decomposition's hegemony [2-3].

In software maintenance, slicing is a method of reverse engineering used to extract a portion of the codes in relation to a specific computation. In 1979, Weiser introduced it with procedural programming for the first time [4]. Slicing was initially designed to make debugging easier, but it has since been found to be useful in a variety of software development lifecycle stages, particularly software maintenance. Numerous of slicing, including union slicing, relevant slicing, and hybrid slicing, have emerged over the past two decades. However, static and dynamic slicing are the main areas of study in slicing. According to Ishio, Kusumoto, and Inoue (2004), program debugging, testing, and verification can be more difficult with aspect-oriented programming (AOP) than with

traditional programming [5]. Slicing might be a more useful method in this area. It can be utilized to identify the model component that impacts or is influenced by the conditions.

Dependence graph (DG) as well as control flow graph (CFG) are the two types of graphs that make up the dependency flow graph (DFG) [6]. DG is a type of directed graph that is typically used to show how different objects depend on one another. A works DG becomes a subset of method dependence graphs that represent a main() method or a program class, as well as some additional arcs that represent transitive interprocedural data dependencies and indirect/direct dependencies among the called as well as a call technique [7]. The dependencies among the concepts of advice, join points, and aspects as well as the constructs that are associated with them were represented by DG in AO. CFG is a model in which each arc—also known as a directed edge—describes how control moves between statements in a program. Each node—also known as a point—represents a statement in the program [8].

## **1.1 History**

In 1968, the NATO Working Conference on Software Engineering coined the term “software engineering” [9]. Even though there are other definitions, these are the ones we use: Software engineering is the process of integrating as well as relating techniques and procedures from a variety of fields, such as engineering, project management, computer science, and the architectural strategy, execution, customization and maintenance of software. The term "software crisis" was used to describe a whole group of issues that were observed in software improvement as well as prompted the launch of the software engineering offensive. Software that did not fulfil the necessities are known as low-quality software, unmanageable projects, and code that was difficult to maintain were all manifestations of the software crisis. Other manifestations included projects that were completed ahead of schedule and on budget. causes of the software disasterEdsgerDijkstra, a software engineering developer, provided the following explanation for the primary reason behind the software crisis [10].

The power of machines has increased by several orders of magnitude! To put it succinctly: Programming was a breeze so long as there were no machineries; Software design was a minor issue when we only had a few weak computers; however, now that we have enormous processors, software design is a correspondingly enormous issue.

Dealing with the software crisis has progressed significantly since the 1960s. Software systems became increasingly complicated to build [11]. However, the exponential increase in sheer

size as well as complications of modern software systems slowed the development of software engineering tools, techniques and conceptions. The goal of making software development an engineering discipline is still, to a large extent, a goal. The existing base of software science and technology is not enough to fulfil the requirements of software implementation now and in the future [12-13].

In 1995, the Standish Group reported that around just 16% of software projects were effective, 53% were plagued by issues (overruns in budget/cost, content insufficiencies), and 31% were canceled [14]. The typical software project was 222 percent behind schedule, 189% over budget, and only carried 61% of the functions specified [13]. Merely 34% of the entire software projects were considered successful, based on the most recent report of Standish Group [15]. There is evidence to suggest that the existing state of software development is far from acceptable standard, despite improvements from 1995 to 2003 [11-13].

Since the beginning, research on software engineering has been guided by two fundamental principles: modularity and the separation of concerns, which are inextricably linked. Software that is constructed in accordance with these principles is easier to manage and comprehend, enhancing software maintenance, progress, and reuse. A software is broken down into its component parts when concerns are separated. A software system's concerns are these pieces, where a concern is a semantically coherent problem domain of interest. A concern could be a necessity like "realtime operation," an aspect in a program like "RSA encryption," a data structure like a B-tree, or even a minor problem such as making a short/long integer and a length counter [16]. The principle of separation of concerns (SOCs) is exemplified by the fact that concerns are the main measures for breaking software down into smaller, more handy, as well as logical components [17].

**Table 1.1-AOP Software Quality Metrics**

<b>Quality Type</b>	<b>Characteristics</b>	<b>Sub-Characteristics</b>
Software Product Quality	Portability	Confirmness
		Replacability
		Installability
		Adaptability
	Maintainability	Modularity
		Testability
		Stability

		Changeability
		Analysis
	Efficiency	Code reduction
		Resource behaviour
		Time behaviour
	Usability	Complexity
		Operability
		Learning ability
		Understanding
	Reliability	Recover
		Fault tolerance
		Maturity
	Functionality	Reusability
		Security
		Compliance
		Interoperability
Accuracy		
Suitability		

However, there is no instruction on how to detect as well as organise concerns in the concept of SOCs. The criterion of cohesiveness proved to be appropriate. The degree to which the pieces of code that address a problem are functionally related is known as cohesion. High cohesion is preferred due to its connection with numerous required software characteristics, such as reusability, understandability, robustness, and reliability. By structuring software in accordance with this standard, software developer is able to focus solely on the problems surrounding a single issue, minimizing the amount of time spent being distracted by the implementation details of other issues [18]. This strategy, according to Parnas, is develop for modification: A programmer organizes software in such a way that the programming that will possibly modify is contained within the implementations of the concerns. By separating concerns, it is possible to modify a concern's execution deprived of influencing or relying on other issues. Modularity becomes the idea that software should be organized into modules or it computes the number of modules that are utilized in a software model.

Structured design, structured programming, program specification and modular programming were all areas of research that led to the development of the concept of modules. It has been decided that the larger system has an essential part as a module and will be able to work with other modules, despite the fact that there are several other definitions. Software modules are self-contained, integrated components. Modularity becomes the result of the SOCs, and a module is a device for implementing a concern [19]. A module (information hiding) offers as well as connects through an interface to conceal definite concerns' specifics. Concern implementations are separated from one another by interfaces, reducing concern interdependencies. An initiating mechanism for the SOCs as well as develop for variation is provided by interfaced modules [20].

The historical backdrop of computer programming as well as programming language study is to a huge degree the historical backdrop of encouraging and further developing division of worries and seclusion [21]. Deliver the appropriate abstractions, languages, frameworks, techniques as well as tools for software developers to build modular and well-structured software presents a challenge for both the industry/research community. A significant step toward resolving the software crisis would be this. Sadly, it turned out to be a challenging endeavor. The goal of this thesis report is to make a contribution to the software field by providing conceptual, methodological, practical, and tool-related methods for enhancing software modularity and concern separation [22]. AOP and OOP, two novel programming and software development paradigms, are the primary subjects of this thesis.

Crosscutting concerns are a subset of design and implementation issues that are the focus of both OOP and AOP. A single implementation/design decision or problem known as a "crosscutting concern" is one whose resolution typically necessitates dispersion among the software system modules, resulting in "code replication" as well as "inter-mingled code" [23]. Crosscutting issues are unique because they pose a trial to established development and programming practices like object-oriented programming (OOP). Crosscutting concerns have been found to result in code that is inherently poorly structured, making software less manageable and understandable [24].

A bad/good software model or programming style does not solve the crosscutting issue. The tyranny of the prevailing breakdown becomes a method of software decomposition that results straight from the absence of support from conventional programming prototypes like OOP. That is, a coding can only be modularized in a single direction (along a single dimension) at a time, as well as various issues, that do not coordinate with that modularization process finished as there replicated form, tangled and dispersed code [25]. This issue is explicitly addressed by OOP and AOP, which

offer concepts for disintegrating software along multiple dimensions. Although modularizing cross-cutting issues is the goal of both OOP and AOP, their approaches to this issue differ. Meta-level language builds that initiate reasoning about as well as manipulating fundamental programs are provided by AOP, whereas OOP is concerned with the automated formation of software from features. A programmer provides the definition about the parts of a program that need to be expanded (also known as join points) as well as a collection of events, additions, or conversions to be carried out at such points [26].

In spite of the fact that it might appear that OOP and AOP are two approaches that are at odds with one another, the findings of this thesis or research reveal that OOP and AOP are techniques that work in conjunction with one another. Different program designs are produced as a result of the various ways in which they decompose and structure software along various dimensions. This research explains how the limitations of AOP and OOP as a pair can be overcome when combined. In order to maximize the merits as well as minimize the drawbacks of both programming paradigms, a synthesis of their strengths and weaknesses are outlined in this research as required.

We need guidelines to help programmers to select the appropriate approach for a particular problem, provided the several individual advantages as well as disadvantages of OOP and AOP [27]. The complete report through this thesis is based on such strategies as well as can be viewed as a historical outline of the author's research into this issue: The evaluation of OOP and AOP served as the basis for the programming guidelines, which led to the idea of combining the two. This work has been assessed in a nontrivial case study as well as contribute to identifying the present usage of OOP and AOP concepts. In a nutshell, the key to integrating, unifying and comparing among OOP and AOP are the guidelines for using them according to their strengths and weaknesses [28]. They direct the way to a deeper comprehension of cross-cutting issues and the consistent execution mechanisms, which, considered as a whole, is a contribution to the discussion of modularity and SOC.

## **1.2 Problem Statement**

Currently, a lot of people would look through the code and talk to a logging method in the right place. The code might only need to be changed in a few places if the system was designed and developed correctly. However, numerous insertions would be required for the majority of systems. We could create a logging class as well as utilize an instance of it to maintain the logging if the system were object-oriented. To deal with a complicated interaction between various databases and

files, we might need to know how classes are arranged in a hierarchy. Because these concerns were intertwined with the code's fundamental functionality, standard language constructs are used to produce a list of codes and classes in AOP. From a maintenance perspective, the system becomes more complex as a result of the recurrent rounds of program conversion as well as assessment.

In order to comprehend the program, numerous studies concentrate on determining the most effective manner to illustrate the aspect code framework. Control flow graphs, dependence graphs, call graphs, and others are some of them. as a means of representation. Using the control flow graph, we can create a straightforward, abstract interpretation-based algorithm that finds possible-path constants without requiring program transformations. However, the algorithm's use of dependence and control flow graphs is related to the program's complexity.

Utilizing a dependence graph presents a distinct challenge. Before we discovered the slicing area, the DGs can only form the data dependency correlation once at the starting of the programming. It cannot simultaneously handle multiple programs and can only represent a single process. None of the algorithms that make use of the various dependence graphs find possible paths, despite their high level of complexity. As a result, a thorough investigation of the dependence graph-based slicing is required. CFGs depict the block-by-block, successive statement-by-statement flow of control from the starting to the end of the program without stopping or the likelihood of branching. Even in complex branches, it can show how processes move through each step. However, it becomes more difficult to remove dead code in low-level code the more branches there are in a program. Because different branches are used and each loop goes through a different number of iterations during execution, it is difficult to extract information from the program's branches and loops.

A perfect program illustration for a DFG would have local implementation semantics from which it would be simple to derive an abstract interpretation. To produce an effective algorithm, it would also be a sparse representation of program dependencies. Both the DG and CFG have their pros and cons. Why not combine the advantages of a CFG and a DG to demonstrate a superior method for representing aspect code's architecture? We demonstrate that an efficient way to depict the framework of aspect code is to use a data structure that can be efficiently navigated for dependence information as well as accurately defined language with local operational semantics. We view it as a specifically defined language with a local functional semantics and as a data structure that can be efficiently traversed for dependence data.

In addition, program slicing has been recommended to evidently extract the entire statements that may be influenced in the aspect-oriented coding for more efficient software maintenance. A step to slice AOPs has been proposed by Ishio, Kusumoto, and Inoue (2002). However, the dependence graph that Ishio's research focuses on cannot be utilized directly. For the purpose of AOP slicing, the concept of a CFG and a DG are combined in this thesis to form a single graph. The design and analysis of various important software engineering metrics will be the primary focus of my research. Additionally, the design as well as assessment of AOP and OOP were performed.

### **1.3 Aims Of This Research**

In Proposed methodology we compare and validate the different metrics of aspect-oriented software empirically [4]. The focus of methodology is to prepare a new metrics for ascertaining the performances of AO [1] software designed in different languages. Proposed work considers other AOP [5] languages such as AspectJ, AspectL and AspectXML [2, 3] and features of these AOP languages can be included in our generic framework. There is a need to develop a tool to get different values of metrics and we will be in process of developing such tool which will help software quality team for quality assessment of the domain.

In my research work, the main emphasis is the systematic analysis of various features and metrics that are important in context of AOSD. The following are the objectives of my research work:

- a) To study the existing literature available in the field of design and analysis of object and aspect oriented metrics.*
- b) To propose a model for measuring the performance of object and aspect software metrics .*
- c) To assess the reusability and maintainability of both paradigm.*
- d) To validate the quality of proposed model for metrics.*

### **1.4 Main Contributions**

The following are the contributions that this thesis makes to the ongoing study of software architecture evaluation:

- a) A technique for assessing and altering software architectures. AOP and OOP were applied to design an employee management system as an iterative approach to transform the software architecture designs. AOP provides a straightforward method for identifying concerns



through clustering of scenarios that are similar. Software architectures can also be transformed using a number of heuristics the architectures.

- b) The process of identifying concerns by clustering scenarios. It is generally agreed that defining software architecture designs benefits from understanding the concerns of employers or managers in an organization. An algorithm for identifying concerns based on clusters of similar scenarios is provided by AOP and OOP.
- c) The mapping, characterization, and measurement of scattering and tangling of concerns and modules. Based on a AOP, the designed employee management system provides a description of the mappings of concerns and modules as well as measures related with metrics such as line of code, difficulty level, volume etc. In order to carry out this characterization, a number of method rules are defined.
- d) The software architecture evolution is depicted by AOPP and OOP methods are useful for keeping track of, how software architectures change.

## **1.5 Research Outline**

The Employee Management System was developed using some of the most effective iterative development techniques. It involves carrying out a set of activities over a number of iterations to develop various project phases. These iterations are time-boxed, and the code for the system that has been tested and is executable from each iteration. Before the next iteration, a single iteration consists of exploring sequences, executing a subset of those sequences using AOP and OOP, measuring metrics, analyzing the outcomes, presenting final conclusions, and improving all tasks. The Employee Management System, which is the subject of Chapter 3, can be implemented progressively with great adaptability to vary, thanks to this strategy. A subset of the 6 well-recognized design sequences are selected iteratively in this research work. The patterns are used to create small software modules for the Employee Management System. The patterns are chosen because they fit the security systems needs and promise to improve with AOP.

The implementation is divided into two approaches for the purpose of developing the security system and assessing its behavior. Using OOP and AOP The application will be coded using OOP in the initial stage. Based on the instructions provided for each pattern, this method applies the patterns as usual [2]. The second step is to use AOP to put the pattern into action. Except for the aspect library, which was made by GregorKiczales and Jan Hannemann [1]. After developing all of the code in this document, the next step is to evaluate both approaches using metrics after the security system has been set up and basic functional tests have been completed.

The metrics' values demonstrate differences between OOP and AOP approaches, allowing one to draw some conclusions regarding the strengths and weaknesses of the patterns chosen. In contrast to conventional OOP implementations, this analysis looks at possible scenarios in which AOP patterns are appropriate or not. These conclusions either confirm or refute the findings of research that asserts improvements in patterns through the use of AOP [3]. The results are then written down and summarized. In order to identify information that are useful for the subsequent iteration, all of the conclusions are gathered. It's important to organize the results before moving on to the next iteration. Preparing the following set of patterns and adding more patterns to the earlier application are helpful. The various patterns of metrics chosen for the Employee Management System are examined in the subsequent chapters. They are collected to connect alike business situations, recognise patterns in associations, as well as draw some metrics analysis conclusions.

New metrics for quality measurements of AOP are the subject of this study. Static metrics have been gathered with open-source tools using it. External quality characteristics are influenced by these static metrics. Depending on definite International Organization for Standardization quality guidelines, the AOP system's quality measurement are be expanded by the integration of additional dynamic metrics. Other software quality structures' theoretical contributions have been deliberated. To identify product quality metrics we are proposing a novel a software product quality for combined software applications (AOP and OOP). The quality of computer code is evaluated with static measurements. For instance, cohesion is the degree to which module components cooperate with one another. There are not various measurements for AO architectures at this time [1], such as coupling and cohesion, SOC, size, and so on. One of the most important qualities for AO architectures is cohesion. The degree of association among modules is measured by coupling. Good design is thought to require low coupling and high cohesion. Based on several research papers [3]-[5], investigators inferred one structure from dynamic element metrics and run-time implementation of software reports. The thesis focuses on element estimations and disregards static, conventional measurements. As metrics have been demonstrated to be pointers of crucial quality features, such as the final system's fault-proneness [4], the significance of software measurement has been progressively renowned by OO software experts. As a result, direct extensions of conventional OO measurements provide the AOP quality pointers. In any case, established coupling measurements are frequently necessary for observational studies of AOP.

## 1.6 AOP Quality Metrics

The module-arranged techniques as well as OO metrics can be evaluated using a variety of models. Researchers provide a variety of programming quality models, [29-32], however, low function is done to evaluate the AO model's quality features. Coupling, complexity, changeability, coupling, reusability, cohesion, maintainability and other qualities are examples of different quality characteristics.

- Coupling is the degree to which the modules are related to one another and form an association with one another.
- Complexity refers to the method for analyzing the code as well as the efforts required to change and modify modules.
- Reusability means using the module again to reduce the amount of coding. Low coupling is required. There are various programming metrics for these characteristics. However, very few measures for an AO system are discussed.
- Maintainability is the variation in a programming point after delivery
- Cohesion is the degree to which the components of the modules are related to one another. Programming measurements serve as a pointer to the nature of a model, i.e., they provide a quantifiable basis. Strong cohesion is appealing.

There are a number of product quality frameworks that suggest methodologies to combine various quality features. Every programming helps you see how few quality components make up the whole. This large picture should help us evaluate the product item as a whole.

The AOP Metrics version 0.3 tool aims to provide a standard measurement tool for perspective situated and article arranged coding. The assignment requires to offer the subsequent outlines:

The organized aspects of the subsequent measurement suite's enhancements:

- Henry and Li's measurements suite, which measures package dependencies;
- Robert Martin's measurements suite, which measures CK;

Classes and aspects can be linked to the actual measurements produced by AOP Metrics tasks. Consequently, classes and aspects will be referred to as modules. Additionally, the task term will demonstrate methods and recommendations [33].

Methods for static investigation extend CK figures metric/class level code measurements in Java (for instance, there is no need for accumulated code). It currently includes a huge collection of measures, consisting the well-known CK.

The engine for developing ASTs with this tool makes use of the JDT Center Library from Overshadowing. The reliabilityversion is currently set to Java 11 [34].

## **1.7 Outline of the Thesis**

This thesis has provided the detailing of work in five chapters that highlights the description of theory, comprehensive literature review on pharmacovigilance, methodology to assess the working of Employee Management System by using both AOP and OOP. The assessment approaches find out the better methodology used for developing an accurate employee management system with comparative analysis, conclusions and future directions. The primary overview of Employee Management System using AOP and OOP is performed with the main contributions of research work and objectives have already been presented in **Chapter - 1**. Further research work has been summarized as follows.

**Chapter – 2**, provides a inclusive literature review on social media assessment of the AOP and OOP with their performance metrics consisting background, origination and modelling schemes with their performance characteristics. The chapter depicts the present methodologies with their main gaps and active research fields. This chapter also discusses the possible introduced methods that can be used to enhance performance. It discusses all the present scenario need to be deal with in order to develop an optimized framework. The chapter can be useful to compare the developed methodology with the existing systems and their performance evaluations.

**Chapter-3**, focuses on the development of employee management system with precise methodology and the selection of significant constraints that were taken into account to assess the AOP and OOP performance as well as their significant variables. It includes the explanation of approaches utilized in computation.

**Chapter-4**, explains the factors and mathematical expressions with results that worked for the two selected systems and find out the results and provides the complete detailing of the better performance approach on employee management system.

Finally, **Chapter-5** includes the conclusions of this research work. It summarizes the major contributions with findings. A discussion has been made and some suggestions are included as the future directions.

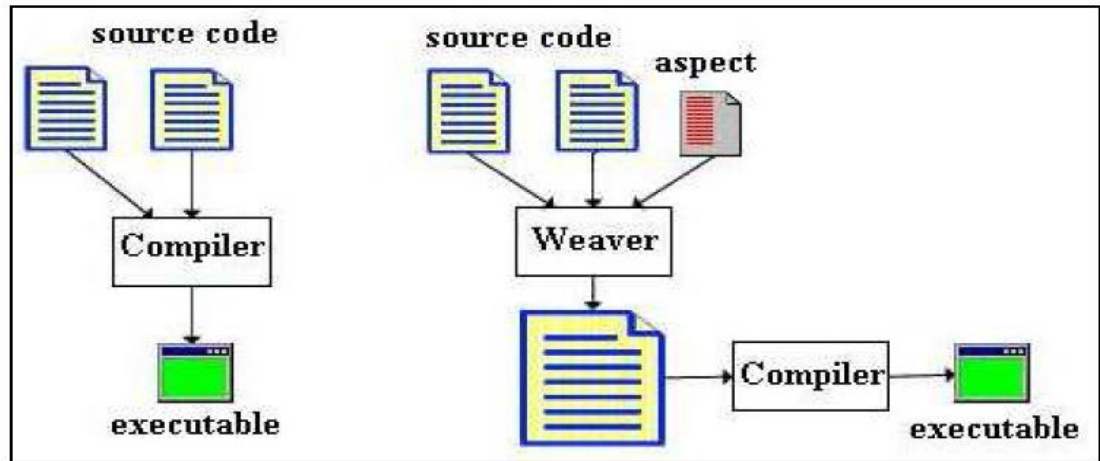
## **CHAPTER 2**

### **LITERATURE REVIEW**

Knowing Aspect-Oriented Programming (AOP) and its slicing based on source code parsing is very helpful. The focus of this thesis is on dependence flow analysis. AOP is a programming style that makes modularity easier by letting crosscutting concerns at the code level be separated into first-class elements, or aspects. Separating the program code into a single unit known as an aspect is what the term "separation of concerns" means [35]. The concern is a particular need or deliberation that must be highlighted to achieve the software systems overall objective. A set of concerns are met by a software system. A crosscut implementation modular unit is an aspect which makes reusable modules out of behaviors that affect multiple classes [36].

AOP is an addition to OOP. It is targeting developers who are dealing with the complexity of OOP development. The OOP program can be dynamically modified by a developer using AOP, resulting in a system that can expand to meet new requirements. Crosscutting concerns can be created independently of the existing OOP code using the AOP code [37]. The most recent program was then rewritten using an aspect weaver from the OOP and AOP combination in a final executable form. The crosscutting and weaving process is shown in Figure 2.1 [38]. Original Object Oriented code can be found on the left, while OOP code with additional aspects can be found on the right.

The original OOP code will not be affected by the addition of aspect code. To restore the original functionality, all we need to do is recompile the code without the component. A unique modularization unit referred as aspect is utilized to execute the crosscutting issues [39]. Rather than combining them in the fundamental modules and repeating them throughout the application, the code can be designed once in an aspect (code tangling). To put it another way, aspects modularize crosscutting issues, making it possible to implement a single (crosscutting) issue within a single module [40].



**Figure 2.1- Representation of Aspect Weaver**

It is very clear how AOP and OOP differ. OOP is the most commonly used approach to address core concerns, while AOP manages the cross-cutting concern. It is not sufficient for many cross-referencing issues, particularly in applications that are complex. The reason for this is that OOP results in an undesirable tight coupling between the variable or method. Policy enforcement, error checking, transaction integrity, multithread safety, security, data persistence, performance, storage management, administration, resource pooling, logging and authentication are all examples of cross-cutting concerns in programming [41].

## **2.1 Aspect Oriented Programming**

A programming model known as Aspect Oriented Programming (AOP) was built toward the end of the 1990s as an alternative to OOP [42]. Diverse efforts to manage concern separation led to the development of AOP. It basically lets software developers work on cross-cutting issues in different processing units to make maintenance and reusability better.

### **2.1.1 Relevant Concepts**

An aspect is the fundamental component of AOP. Avoiding scattering and tangling effects, an aspect separates cross-cutting concerns in definite locations of a code [43]. Advice, pointcuts and join-points are the fundamental components of an aspect [44].

A specific location in the source code is called a join-point. It ought to be any interesting place in the code, like an exception handler, a constructor, or a method call [45]. The official statement of a join point within an aspect is known as a point-cut. When the code matches the join-point, it sets all the rules [46]. The ability to match a large number of join points in a single point cut is regarded as an efficient aspect in AOP [16], making it possible to access many portions of the

code in one location. The program is implemented when the point-cut is attained as an advice. There are three types of suggestions: before, after, and in between. The point-cut is reached either before or after the before/after advice is activated. Because it is implemented rather than the original point-cut code, around-advice is more complicated [17].

A novel strategy for dealing with issues that span multiple domains is provided by AOP. Several domains of software engineering, including system modelling, framework, progress, and testing [21], see it as a promising technology. In order to progress AOP as a technique as well as develop novel domains of research with it, researchers are working on several subjects [23, 24]. In both the business world and academia, a wide range of projects have utilized this paradigm, resulting in a variety of outcomes [9].

### **2.1.2 Development Process with Aspects**

The stages that follow provide the briefing of how aspects might be utilized in a software improvement procedure; however, there are a variety of ways to do this [47].

First and foremost, it is essential to determine which software restrictions are regarded as cross-cutting issues [16]. For instance, in the event that a software requirement states, "develop a log to save the alterations in the values of entire object features," Any call to the "set" operations ought to be recorded, it stands to reason. This method of code analysis reveals potential AOP join-points [18]. The subsequent stage is to develop an aspect as well as affirm the point-cuts with their corresponding advice after the join-points have been identified [16]. The weaving process concludes after all aspects are created. The application code is mixed with the aspects during this procedure. This procedure is usually carried out at compile time, but it can be carried out at various stages [16]. Business and aspect logic are combined to create binary files following compilation. These files are prepared for execution at runtime. The most common steps for developing applications with AOP are the previous ones. However, the implementation of a programming language may alter these steps. There are numerous programming languages, ranging in maturity [5].

## **2.2 The AO Paradigm and Software Development**

The briefing of pertinent studies on AOP and design patterns is provided in this section. It shows how AOP is altering the methods of software progress is performed as well as how it could change how design patterns are used.



### 2.2.1 AOP and Software Development Process

The software engineering community has used AOP in a number of projects and has attained a substantial maturity level [9]. The following illustrates the growing impact of features in various software progress domains [27, 28].

### 2.2.2 Development

New software application development methods are promoted by AOP in some studies [21, 29]. To identify requirements with cross-cutting concerns, those methods necessitate adding additional steps to the software development procedure [19]. Using point-cuts and advice appropriately, it is essential to develop an appropriate execution with AOP to fulfill those necessities [30].

Aspects may have an impact on software component development, according to additional research. According to the findings of that study, adaptability of third-party software components can be enhanced using AOP. Furthermore, it permits software component extensions and modifications without changing the source code [28].

### 2.2.3 Testing

According to investigators, the novel methods are required to find, develop as well as run test cases for AOP-built applications. Cross-cutting issues typically necessitate distinct test cases, which are even more so when they are executed using AOP [22].

In addition, it becomes important to run the aspects in various types of testingsystem, incorporation, and unitsince their integration with the code occurs at compilation timethe weaving procedureas well as it becomes essential to ensure that this combination was successful [22].

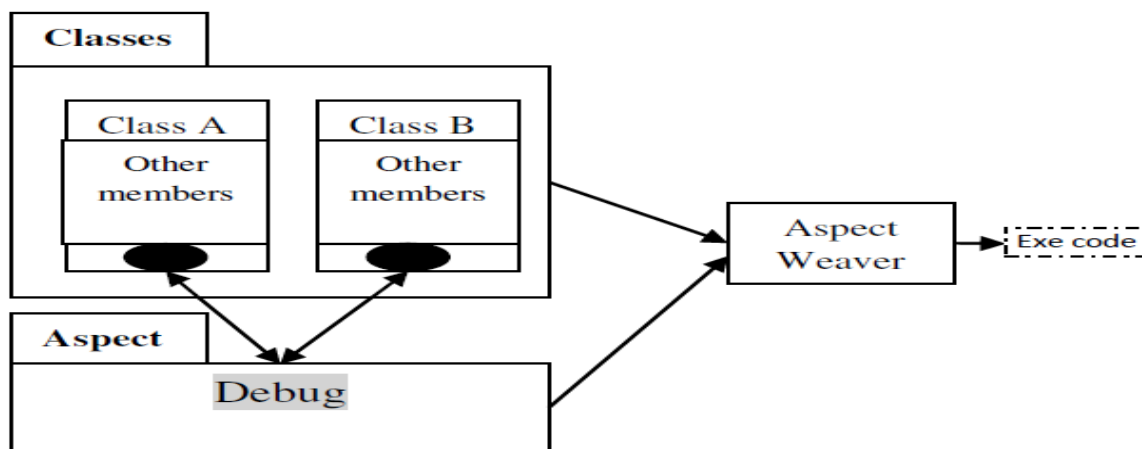


Figure 2.2 -Weaving in AOP

## 2.2.4 AOP and Unified Modelling Language

Modeling languages are another AOP research domain [31]. Object-Oriented modeling has been represented by the well-known Unified Modelling Language (UML), which is used in both the software industry and academia [1]. Extensions of the UML notation are being used by researchers in a variety of techniques to illustrate the complications of AO system [32, 33]. With these extensions, software developers can use a variety of UML diagrams to represent various AOP features [34, 35]. Based on the AOP execution, report, code generation, support, of tool, and other aspects, few methodologies offer superior properties to model aspects than others [36]. It has the profile of UML that is appropriate for the execution of AspectJ, according to a group of researchers [37]. UML stereotypes are used to represent aspects, point-cuts, and advice in this profile. The Security System's logging requirement to provide an explanation of the relevant aspects of this UML profile.

As a UML class, the aspect Logging Operation is created. It is above the name and contains a stereotype called "aspect." Software developers are able to identify and distinguish aspects from domain classes thanks to this stereotype. All of the point-cuts/advice are depicted as functions, each with its own set of stereotypes. The stereotype "pointcut" appears before the name of the point-cut "personDetectedChanges."

Similar to one another, the prefix "advice" appears before the names of both advice declarations. In addition, the appropriate modifiers indicate whether they are carried out prior to or after the point-cut is attained. A stereotype called "crosscut" is used to represent a relationship with the class Room Sensor in UML. This depicts that the aspect utilizes that class as well as involves cross-cutting concerns that are associated to the class Room Sensor.

## 2.3 Object Oriented Programming

Researchers [1,11] defines OOP as the procedure of detecting objects with their attributes, determining which operations every object requires, as well as creating interfaces among objects. There are three steps involved in class design: first, object definition, then object data members, and finally, object communication. As a result, class design is more abstract than the standard data or procedural approach. OOD differs from conventional procedural design due to the task of class design.

The following are some fundamental terms that are frequently used in object-oriented metrics:

- a. *Object*: A state-preserving entity with a variety of operations to investigate or alter such phase is an object.
- b. *Message*: It becomes an appeal for an object to perform an operation on another object.
- c. *Class*: A objects collection with a similar structure and behavior that are shown by a collection of techniques. It can be used as a template to create an object.
- d. *Method*: An operation on an object that is accessible to all classes does not have to be unique.
- e. *Instantiation*: The procedure of binding or integrating the definite info to an object instance.
- f. *Inheritance*: A connection between classes in which an object in one class gains features from one or more other classes.
- g. *Cohesion*: The degree to which a class's methods are connected to one another.
- h. *Coupling*: If A sends a message to B, then Object A is coupled to Object B.

A subset of software metrics as the software quality metrics can concentrate on the project, process, as well as product's quality features. In general, project metrics are more nearly associated to software quality metrics than process and product metrics. Nevertheless, the quality of the product is definitely influenced by project parameters like the developer number, their organization framework, size, schedule and skill levels.

Berard examines the unique position that object-oriented metrics hold in software metrics research [48]. He identifies five distinctions between OO metrics:

- Object abstraction techniques
- Inheritance
- Information hiding
- Encapsulation
- Localization

The author asserts in the article's opening section: "... for a specific procedure, product, or person, 3 to 5 suitable attributes observed to be a applied upper limit, i.e., further metrics (above 5) do not generally offer a substantial return on investment," states that software engineering metrics are rarely useful on their own.

One of the most significant works on OO metrics is OOP Measurement. Whitmire treats patients with extreme rigor: establishing the theoretic fundamentals, contextualizing measurement, as well as taking design features via its metrics. Volatility, similarity, primitiveness, cohesion, completeness, sufficiency, coupling, complexity and size are all covered by Whitmire's metrics [49]. Motives and ancestors, empirical perspectives, formal properties, empirical relational structures, potential measures, and so forth, within each area are talked about. The author offers a novel perspective on numerous software measurement issues. The development of a solid scientific architecture for comprehending as well as evaluating till date most time consuming perspective of software progress framework is Whitmire's most significant contribution. The book's heavy emphasis on rigor also has a disadvantage. Practitioners rarely have the time to fully comprehend and apply Whitmire's findings because of the arduous nature of industrial software development.

## **2.4 Design Patterns**

It becomes asserted that AOP addresses the challenges raised by design sequences in the prior illustration. Aspects move the invasive pattern code to a central location [13], thereby preventing tangling as well as dispersing in the program [10] as well as enhancing the patterns' locality/reusability [14]. AOP makes it possible for features to add classes' features, functions, and even interfaces without changing the source code of classes [3]. Consequently, multiple inheritance in pattern code is eliminated [12].

On the other hand, researchers have identified potential problems as well as enhancements for AOP [38]. In complex patterns, the aspect code can sometimes have unanticipated effects. It makes it harder to understand the patterns and the more classes that participate in them, including aspects [39]. The trade-offs between the AOP/OOP methodologies are the focus of this thesis in general. The next chapter provides specifics about the project, including its primary goals and the methodology that was used, depending on the data that is depicted in this chapter.

Some problems with the patterns might be solved by using AOP to put them into action [12]. Conversely, such kind of execution results in interesting side effects in the applications of software [39]. Our research aims to use AOP to execute a pattern subset as well as determine conditions in which this integration is or is not appropriate. A program is created to put a variety of patterns into action. Chapter 3 provides a description of the selected application. This system was created throughout the document, adapting to new requirements. The system is more complicated and has a diversity of patterns at the end. When contrasting the OOP method with the AOP method,

it is helpful to highlight its advantages and disadvantages for patterns. Metrics that have been used to evaluate the characteristics of the pattern serve as the basis for the conclusions.

## 2.5 Metrics

Metrics are required in order to evaluate the OOP and AOP pattern implementations [40]. Specific metrics have been used in studies to evaluate these programming paradigms. Software developers can analyze features in terms of GRASP concepts [1] and cross-cutting concerns [41] with the help of these metrics. A brief explanation of each attribute, as well as the metrics and methods for measuring them, can be found in Table 2.1.

**Table 2.1 - Existing Studies Towards Software Metrics [50]**

<b>Researchers</b>	<b>Performance parameters</b>	<b>Challenges</b>	<b>Approaches</b>
Anna et al.	Diverse domain, complexity degrees	To compute the maintainability/reusability degrees of AO models.	Quantitative/Empirical assessment, Aspect-oriented software development (AOSD).
Basili et al.	C&K OO metrics provides superior against traditional metrics.	Investigated the suitability of OOP suggested metrics depicted by Chidamber&Kemerer.	Empirical authentication
Kaur et al.		Analysis/classification of several reusability attributes.	K-Nearest Neighbor-based approach.
Kaur et al.	Show an effective system targeted for software coders.	To discovering structural aspects for software constituents.	Software metrics utilizing neural networks.
Wu et al.	Reliability/Reusability.	To perform comparative assessment on Java, C# and C++ codes by utilizing OO Metrics.	In software engineering, comparative assessment on software metric reusability.
Linda et al.	Reliability, software quality.	To assess various metric suites for OO schemes.	Progress of a software framework such as “Class Break point Analyzer (CBA)”
Scotto et al.		To assess the metrics efficiency.	Web Metrics for SQL queries.
Shaik et al.	The effort,	To maintenance/itemize of software.	OOP
Liu et al.	different metrics	The gap among design and quality measurement of metrics.	OOP.

Alcalá et al.	Complexity and safety.	Enhancing the flexibility of model structure and performance.	Linguistic framework utilizing weighted double-consequent fuzzy instructions.
---------------	------------------------	---	---

*Separation of Concerns:* This metric evaluates the degree of concern disaggregation in various portions of the code, for example, operations/classes in OOP as well as advice/aspects in AOP [43].

*Coupling:* The complexity of the classes' interactions is the subject of this OOP feature [1]. Coupling measures the rigidity and complexity of a class's relationships and the number of dependencies on other classes [42].

*Cohesion:* Cohesion, in OOP, is the correct assignment of a class's responsibilities [43]. A class with a lot of cohesion only does specific things that are related to its purpose and doesn't do a lot of things [1].

*Size:* This generally computes the size of classes as the number of implementable lines of code (lines of code without empty lines or comments) [41].

The OOP technique relies on OO Metrics that will be beneficial during the software application framework and development stages.

Metrics can be broken down into a variety of groups, such as metrics for the evaluation system, metrics for the design framework, attributes for the metrics for quality assurance/testing and source code, and so on. OO Static and Dynamic metrics make up the majority of OO metrics.

Software static artifacts like specification documents, design diagrams, and code listings are used to collect static metrics. Coupling Between Objects (CBO), Weighted Methods per Class (WMC) and Lines of Code (LOC) are just a few examples. Chidamber et al. [51] utilize runtime software behavior data to evaluate dynamic metrics. Examples include dynamic coupling between objects, dynamic lack of cohesion, and so on. [Mitchel et al.'s [52] object oriented static metrics are design metrics used to assess the software system's quality using static analysis data from object-oriented systems. Various researchers have proposed a variety of object-oriented metrics.

The majority of referred metrics are Chidamber and Kemerer (CK Metrics) [51]. Reseachers have defined NOC, DIT, CBO, LCOM, RFC and WMC as six metrics. These metrics are used to

assess the complication in association to the quality factors of reliability, functionality, maintainability and usability. The objective of each of such metrics is to assess the OO application's design instead of the system's execution.

Researchers have provided attributes for static aspects of software implementation that are influenced by Class Internal, Class Inheritance, and Class Size, with a prominence on counts of reusability of operations, operations, attributes in the cohesion oriented in class Internal, inheritance hierarchy, and so on,

The MOOD metrics that researchers which are used to assess the design of OOP design approaches such as Polymorphism Factor (POF) and Coupling Factor (COF), Method Inheritance Factor (MIF), Method Hiding Factor (MHF), Attribute Inheritance Factor (AIF) and Attribute Hiding Factor (AHF) [53].

Researchers gave Quality Model for the Object Oriented Design (QMOOD) metrics name like- Design Size of Class (DSC), Number of Hierarchies (NOH), Number of Polymorphic Methods (NOP), Number of Methods (NOM), Measure of Functional Abstraction (MFA), Direct Class Coupling (DCC), Class Interface Size (CIS), Measure of Aggregation (MOA), Data Access Metric (DAM), Cohesion Among Methods of Class (CAM), and Average Number of Ancestors (ANA) are all part of the Quality Model for Object-Oriented [54].

Many measurements apply to aspects and classes [55] both of these modularization units will be referred to as modules in the following discussion. In a similar vein, the term "operation" encompasses both aspects, advices and class methods. Therefore, these are few aspirant aspect metrics. Depth of Inheritance Tree or DIT-length of the longest route that leads from a given module to the root of the aspect hierarchy. The greater an aspect's depth in the hierarchy, the more functions it may inherit, making it more difficult to comprehend and modify [56].

Size and difficulty: The class's size in terms of methods and attributes is measured by the numbers NOM (Number of Methods) and NOA (Number of Attributes). CC (Class Complexity) and WMC (Weight Method Complexity) are associated to classes as well as utilized to compute total complexity by varying the total number of methods/operations. Such measurements are needed to have low features due to the fact that classes are intended to be developed as succinctly as possible [11].

*Cohesion:* Four class-level measurements are used to measure cohesion and are calculated in a variety of ways to show how part functions and methods work together. LCOM (Lack of Method Cohesion), ICH (Information Based Cohesion), LCC (Loss Class Cohesions), and TCC (Tight Class Cohesion), are the four levels [12].

*Reusability:* Framework-level reusability metrics such as the SR (Specification Ratio) and RR (Reuse Ratio) are both available. Separately, the proportions of subclasses to all classes as well as superclasses are used to calculate them. Wide-ranging reusability metrics are anticipated because classes are expected to be extensively reused [13].

*Polymorphism:* Polymorphism measurements at various levels include the PF (Polymorphism Factor) and NMO (Number of Methods Overridden by the Class). To be specific, NMO is a metric at the class level that computes the number of approaches overridden by a single subclass, whereas PF is a metric at the architectural level that computes the degree of method overriding throughout the model [14].

*Encapsulation:* The indicators AHF (Attribute Hiding Factor) and MHF (Method Hiding Factor) demonstrate the degree to which attributes/methods are effectively concealed within classes. At the architectural level, these calculations are taken [15].

*Coupling:* From various perspectives, class coupling is evaluated using five measurements. At the framework level, the CF (Coupling Factor) metric is utilized to evaluate the coupling of all classes. The other four measurements, which can be examined, compute coupling at the class level. MPC (Message Passing Coupling) and RFC (Response for the Class) are two of these measurements that are used to examine technique coupling; DAC (Data Abstract Coupling) encapsulates information coupling between classes; and CBO (Coupling Between Objects) is a measurement that shows coupling among class occurrences [16].

## **2.6 AOP Languages and Constructs**

The inherent heterogeneity of AO languages and mechanisms is one of the concerns why studying the influence of AOP on maintainability is problematic. Different encapsulation and composition mechanisms are utilized by various AOP languages, which typically incarnate distinct AOP blends. Other programming frameworks, like subject-oriented programming (HyperJ), feature-oriented programming (CaesarJ) and collaboration languages (CaesarJ), may also provide them with abstractions and composition mechanisms. Although the majority of AOP languages incorporate



traditional AOP characteristics like aspects, advice and joinpoint models, or equivalents, every language has its own distinct characteristics that make cross-language evaluation challenging.

Ten of these features are listed in Table 2.2 for AspectJ, HyperJ, and CaesarJ, which are three of the most widely used AOP languages. For example, advanced dynamic pointcut designators like "cflow" are supported by AspectJ. HyperJ modularizes both crosscutting and non-crosscutting behavior through the use of hyperspace modules. As a result, HyperJ does not explicitly differentiate among classes and aspects like AspectJ does. Composition Relationships are one of the additional HyperJ-exclusive abstractions. These define how the surrounding modules should be accumulated using merge-like operators. Lastly, CaesarJ encourages the utilization of virtual classes to execute crosscutting behavior that is easier to plug in. Aspect Collaboration Interfaces connect this pluggable behavior to the base code.

**Table 2.2 - AO Abstractions and Mechanisms Unique to Three Main AOP Languages**

<b>AO Language</b>	<b>Mechanism/Abstraction</b>
AspectJ	Aspect
	Dynamic Pointcut Designators
	Intertype Declaration
CaesarJ	Aspect Collaboration Interface
	Virtual Class
	Weavlet
HyperJ	Concern Mapping
	Hyperspace
	Composition Relationship
	Hypermodule

One of the most widely used AOP for Java implementations is AspectJ [16]. It was developed by Gregor Kiczales, the principal researcher on AOP, and XPARC (Xerox Palo Alto Research Center). According to recent research, AspectJ becomes one of the most advanced AOP executions. It continually offers enhancements as connected with the tools, stability, documentation, as well as core language characteristics [5]. The project is being designed using AspectJ, which is also regarded as an execution with scalability features, based on these facts. Software developers can successfully apply this feature to large-scale systems thanks to this feature. It contributes to the

meaningful reduction in scattered code while having a minimal impact on performance. These outcomes demonstrate that AspectJ as an implementation of AOP is mature [44].

Using the following procedure, AspectJ works: First, the application's business logic-reflecting Java code is created. Second, components like pointcuts, marker interfaces, and advice are designed separately in distinct units (source code files). The weaving process follows the development of all aspects and business logic. The AspectJ compiler, which compiles Java source code, finds point-cuts in it, and mixes Java code with aspect code [4], accomplishes this. In the end, the compiler generates Java runtime files that combine aspect logic as well as business logic [19].

The majority of Integrated Development Environments (IDEs) have AspectJ implementations. Developers can use those implementations' tools to properly work with Java code and AOP. The applications for this project were developed with the help of the AspectJ Development Tools for Eclipse 2.2, Eclipse Indigo IDE 3.7, the Java Development Kit version 1.6.0 and AspectJ 1.7.0,. These tools provide features like cross-references between Java code as well as aspect code, which is one advantage of using IDEs [45].

## **2.7 Fundamental Concepts in AOP**

We know from the previous discussion that the fundamental idea of AOP helps us modularize cross-cutting issues. We can incorporate the following ideas into a generic model in order to comprehend the fundamental idea in detail and the fundamental characteristics of AOP systems:

- Identifiable points during system execution—The system exposes points during system execution. Method execution, object creation, and exception throwing are all examples of these. Join points are system points that can be identified. Take note that join points are points during the implementation of a system as well as are present in all systems, even those that do not employ AOP. These points are simply identified and categorized by AOP.
- A method for choosing join points: Executing a crosscutting concern necessitates choosing a particular join points set. A complicated selection may be formed by combining two pointcuts. Context can also be gathered at the selected points with pointcuts. A point-cut, for instance, might gather method arguments as context. An AOP system's join point system is made up of the idea of join points and the pointcut construct.
- Constructs to alter the system's static structure Sometimes, we need to alter the system's static structure in order to effectively implement cross-cutting functionality. We may, for

instance, require to include the logger field in every traced class when implementing tracing; Such modifications are possible thanks to inter-type declaration constructs. Before the system can be put into action, we may require to find definite situations, usually the existence of certain join points; Such options are made possible by constructs for weave-time declarations. Because of their impact on the static structure rather than the system's dynamic behavior, all of such concepts are collectively represented to as static crosscutting.

In an AOP system, Figure 2.2 depicts all of these players and their relationships to one another. A portion of the model can be implemented by each AOP system. Spring AOP, for instance, places an emphasis on its runtime nature, which is why it does not use weave-time declarations. However, the join point model is so essential to AOP that it must be supported by every AOP system; everything else revolves around it.

We employ AspectJ as our language tool to represent our fundamental concept of a dependence flow graph. Because it is one of the most widely used AOP languages, AspectJ was chosen. The most widely used typed aspect language for Java is AspectJ, which was created at PARC (formerly Xerox PARC). On standard JVMs, AspectJ-written programs run. In standard Java, the classes are developed as well as programmed independently of the crosscutting code. The crosscutting code is enclosed within aspects.

The structure of an aspect in AspectJ is described in the following. To put it another way, it provides a succinct explanation of basic constructs (aspect constructs) or components, which are the construction blocks that make up the modules that express the execution of the broader concern. Take note of the fact that the AOP terminology has not yet been standardized at this point. Although their precise meanings can vary between systems and personal perceptions, the terms weaver, join point, and aspect are used as defacto standards.

**Join point:** A well-defined point in the core code (or base code) where a problem will intersect the application is known as a join point. Where horizontal bundling happens in the code, a distinct set of well-defined points in the program flow—previously discussed to as markers or join points—are required. The compiler is able to recognize 11 different kinds of join points that are defined in AspectJ. Method calls, exception execution, field assignment, and so on are among them. The code that follows provides an illustration of a join point.

```
public class Account {  
  
    static float balance;  
  
    .....  
  
    void credit(float amount){  
  
        balance +=amount;  
  
    }  
  
}
```

Access to the balance instance member (set a member) and the execution of the credit() method are two of the Account class's join points. The precise syntax will differ from language to language, but the join point's purpose is to match the program's well-defined implementation points. Requirements engineering, architectural modeling, and design, among other earlier stages of the software development process, have recently begun to incorporate AO concepts. Aspect-Oriented Modeling (AOM), which focuses on approaches at the design level, supports the SOC at the development level as well as the capability to efficiently address the complication of creating software that addresses interdependent concerns.

## **2.8 Maintenance of Aspect-Oriented Programs**

This thesis work try to narrow our scope of research based on the discussion that came before it. This research draw comparisons from other papers that are related. Research is interested in the graph-based implementation of a maintenance aspect-oriented program. An inter-procedural aspect control basic system is presented by Bernardi and di Lucca (2007) [57]. The system's interactions are proposed to be represented by a software flow graph using non-weaving aspect code. Using the Inter-procedural Aspect Control Flow Graph (IACFG), the proposed research depicts an AOP system and describes how aspects interact with program components. As their starting point, they studied Zhao's work (Zhao, 2002) [58]. The idea was to make it simpler to identify how aspects and the base code structure affect one another. They determine the IACFG customization of the CFG graph and present the findings using a tested code. However, there is still room for development in the graph's precision when it comes to interceptions and polymorphic calls.

Researchers provides a quantitative evaluation of the constructive as well as destructive impacts of AOP on usual Web application maintenance tasks. They offer two distinct approaches to application development for the same application. In order to determine, which of the two versions is easier to maintain; one is object-oriented and the other is aspect-oriented. The analysis of fundamental modularity characteristics like coupling, cohesion, conciseness, and concern separation is driven by them. Their experiment covered six aspect components and 35 OO components. The utilization of AOP resulted in fewer LOC, improved concern separation, and components with lower internal complexity and weaker coupling, according to their findings. However, the experiment was designed with the based and AO codes separated.

The solution of separating the AO and OO programs to identify a suitable region for maintenance activities was presented in both papers. A graph representation was proposed by Bernardi and di Lucca (2007) as a means of simplifying AO maintenance [57]. Researchers suggested OO and AO to create a single alternative web application. However, the primary focus of their research is on determining how aspect-oriented abstractions support concern separation. A new strategy for maximizing parallelism in such codes was presented by investigators. The kinetic dependence graph is the data structure on which this strategy is based. It consists of a dependency and is incrementally updated whenever a task is accomplished to reflect variations in the dependence construction. The method lets programmers write such applications with a high level of abstraction, a runtime, and parallel execution.

Table 2.3 provides a concise summary of the reviewed papers. The work's reference number is listed in the first column. The research's title can be found in the second column. The researcher's method of analysis is listed in the third column, followed by their method of slicing in the study. The programming language they use for their implementation is listed in the final column [59]. The corresponding work, as shown in Table 2.3, involved analysis and slicing for a variety of purposes, including maintenance, testing, debugging, and so on.

Numerous researchers classify the type of maintenance activity in order to accomplish the objectives of software maintenance. The first known researchers to propose categories for software maintenance were Lientz and Swanson (1980) [60]. Software maintenance was previously broken down into the following five categories: Preventive maintenance, maintenance repair, maintenance integrity, maintenance adaptability, and maintenance evolution. Additionally, the international consensus on maintenance work categories has been documented by the IEEE (IEEE Standard). 1219-1998, 1998) were divided into four categories (Table 2.3).

**Table 2.3- Categories of Software Maintenance (IEEE Std. 1219-1998)**

	<b>Correction</b>	<b>Enhancement</b>
<b>Perfective</b>	Adaptive	Perfective
<b>Reactive</b>	Corrective	Adaptive

However, the International Standard Organization (ISO/IEC Standard 14764, 2000) has recently modified five categories, as shown in Table 2.4. User support has been added to the original IEEE categories. This standard served as the foundation for a lot of software maintenance researchers' work.

**Table 2.4 -Categories of Software Maintenance**

<b>Category</b>	<b>Description</b>
Adaptive	An activity of software maintenance designed to adapt to a variety of conditions.
Preventive	A maintenance activity aimed at preventing issues before they arise and enhancing future maintenance and enhancement.
Corrective	A routine maintenance task to correct errors discovered after software development is finished.
Perfective	A maintenance activity that was carried out to enhance the system's performance, maintainability, or other characteristics.
User Support	An activity in maintenance that is not perfective, preventive, corrective or adaptive but instead responds to user demands.

## **2.9 Current Research Work on AOP**

This section explains the pertinent metrics utilized to analyze empirical studies as well as maintainability that were performed to define the efficacy of several maintainability attributes.

### **2.9.1 Maintainability Metrics**

The majority of AOP maintainability attributes were resulting from OO systems' maintainability metrics [8]. However, in some instances, it is necessary to introduce new metrics that take into account AOP's unique characteristics. For instance, the authors of [8] have provided AOP-specific specialized metrics for maintainability. The various studies and metrics used to evaluate AOP software's maintainability are compiled in Table 2.5.

## 2.9.2 Empirical Studies

Investigators carried out research to demonstrate how AOP can be utilized to enhance the maintainability of COTS-based commercial models [61-62]. According to their research, it will be difficult to maintain the system if the program to call the COTS libraries is dispersed throughout the glue code. They demonstrated that the COTS-based model can be simply handled if the glue code is constructed utilizing AOP. In their experiments, they made use of the Java Email Server. They demonstrated that when AOP is used in the glue code, the amount of code that needs to be changed is minimal, which makes COTS-based models more adaptable. In their research, they used metrics based on size.

Researchers conducted research to examine the framework stability of an application constructed with software constituents as well as features in the face of alterations. In their research work, the constituents are referred to as COSMOS (Component Service Model with Semantics) components [63]. They made the comparison among eight releases of four distinct MobileMedia application versions in their research: a component-based version, an OO version, an AOP version, a component-based version, and a hybrid version that used aspects as well as components. The MobileMedia application's first two versions were already in existence, and the final two were refactored versions of the first two. They computed the influence of alteration in their study by counting the constituent operations/number that were altered, integrated to, or eliminated. According to their research, the hybrid version required fewer modifications than the other versions.

In another work, investigators conducted research on the impact of change on AOP systems [64]. They utilized AOP models that had been refactored from their OO versions for their research. Researchers reduced the number of modules in the models they utilized from 149 OO modules to 129. Classes are referred to as modules in the AOP/OO versions, while aspects and classes are referred to as modules in the OO and AOP versions. The metrics as well as the tool for gathering metrics information as defined in [64], were used in their research. According to their research, AOP systems have a lower impact on change than OO systems. Additionally, they discovered that if non-crosscutting concerns are moved to aspects, the change will have a greater impact on these modules. Their evaluation was carried out at the module level because they based their assessment of the AOP's maintainability on the system's changeability.

Using the GQM method, investigators compared among AOP and OO- modularity systems. She utilized a number of systems with AOP as well as OO [65]. Such models had also been utilized in a number of other work to investigate the impact of AOP. CBO and LCOM metrics were used to

compare the modularity. Pet Store, Telestrada, Elmp, the CVS Core Eclipse plug-in, JHotDraw, Health Watcher, Prevayler, HyperCast, HyperSQL and Prevayler, Berkely DB Database were the systems that they used in their research. They discovered in their research that AOP does not provide any advantages as modularity.

**Table 2.5 - Summary of Maintainability Metrics.**

<b>Research work</b>	<b>Metrics Tested</b>	<b>Study Dependent variable</b>	<b>Briefing</b>
Burrows et al.	All Ceccato and Tonella	Fault-proneness	Investigatoresoffered a novel metric base aspect coupling (BAC), that comutes the coupling among aspect as well as base class. This work depicted that the two attributes that shown the robust relationships to errors were BAC as well as CDA
Eaddy et al.	CDO, CDC, DOSM, DOSC	Fault-proneness	Analyzed the relationship among crosscutting concerns and faults. They discovered that the more dispersed a concern is the more errors in its execution are. Concern metrics utilized to estimate the scattering of a concern. Such metrics are not dependent on the program size
Kumar et al.	WOM	Changeability	Analyzed the relationship among the WOM metric as well aschangeability. They discovered that the WOM can be utilized as a weak indicator of maintainability. When compared to OO systems, the impact of development in AOP systems is lower. The number of modules changed was used as a measurement of maintenance effort.
Kulesza et al. [12]	Sant'Anna metrics that involves VS, WOC, LCOO	SOCs, Coupling, cohesion	Since an increase in these metrics was always accompanied by a decrease in development effort, they cannot be used as predictors of maintainability.Insufficiently conclusiveLCOO maintenanceability metric.
Shen et al.	Ceccato and Tonella	Changeability (maintenance tasks and coupling)	Metrics that are linked to maintainability.



Przybyek	metrics for coupling (CDA, CAE, RFM, CMC and CFA)	Modularity	The modularity of a system was measured using CBO and LCOM. Cohesion and aggregate coupling should not be taken into account because coupling should be measured independently of the system's number of modules..
----------	---	------------	--

Lippert and Lopes [66] found that the JWAN framework's exceptional handling code was lessened by a factor of four after using AspectJ to refactor it. The JWAN architecture was constructed using the proposal by contract method. As a result, these contracts were the focus of the refactoring. In the JWAN framework, for instance, every method that returns an object must ensure that it is not null. This becomes an accurate illustration of an aspect that could be executed more effectively using features. Additionally, the JWAN framework's exception handling was a main design characteristic because approximately 11% of the programming was geared toward exception handling. Likewise to the previous research work, ours is similar.

Although, by evaluating the various maintainability characteristics, we wish to expand the scope of our investigation. For instance, the authors of [11, 15] concentrated solely on the ability to change. Researchers' investigation [16] looked at the number of components or operations that were changed, integrated, or eliminated to determine the impact of the alteration. The focus of investigators' research [14] was on determining how modular OO and AOP systems are. By involving few extra metrics for assessing the COTS-based model's maintainability, our research work can complement these other research works. Cohesion, complexity, and coupling are examples of these metrics. The majority of such metrics concentrate on the various features of maintainability. Additionally, we wish to examine maintainability on two levels: structural level and concern level. We want to see how well they measure maintainability by doing this.

## 2.10 Deeper Perception on Software Metric Development

Baker et al. depict a serious perspective on the condition of software computations while jokingly referring to themselves as the "Grubstake Group." "... if there occurs a rigorous/formal base for software measurement," the authors are persuaded, is the only way to establish an environment for software measures [67]. The people who define, proof, and offer tool support for the software procedures will need to understand this foundation, not the software measure users. The work applies formal measurement theory concepts to software metrics and emphasizes the necessity of identifying and defining: characteristics of software products and methods.

- Abstractions or formal models that represent the characteristics.
- Important relationships and orders that are established by the attributes of the models and exist between the objects that are being modeled.
- Number system mappings from the models that keep the order relationships.

Additionally, the authors emphasize the importance of sound validation schemes in determining a software measure's reliability and lament "...a common deficiency of validation of software procedures." In conclusion, the paper demonstrates that a measurement theory framework should and can be used to develop software metrics.

Three measures of software design complexity—design, data and structural complexities—have been defined by Card and Glass [68]. The fan-out concept is used in the structural and design complexity measures to indicate the module number that are straightly raised by a module and are immediately subordinate to it. The sum of the data complexity as well as structural of a system is its complexity. The authors speculate that as each of these complexity levels rise, so does the system's overall architectural complexity, necessitating greater integration and testing efforts.

The same as Armour has described the experience of executing software aspects I initiatives at Motorola in his statement of starting a metrics program at their association [69]. The researchers conclude that metrics can reveal the domains in need of development based on the practical difficulties encountered during implementation. The actions taken in response to the findings of the metrics data analysis are the only thing that will determine whether or not actual improvement occurs. This research emphasizes the crucial realization that metrics are merely means to an end; Through measurement, analysis, and feedback, improvement can ultimately achieve its ultimate objective.

Armour continues the discussion in his previous book on implementing metrics in a large organization by highlighting the dual advantages of utilizing metrics, which include faster project management and improved processes [69]. Grady begins by discussing the strategic use of software metrics in project management before moving on to discuss the strategic aspects of process improvement. In a chapter titled "Software Metrics Etiquette," that has many durable messages, including the fact that metrics are not intended to measure individuals, the book provides a unusual understanding into the human problems associated with relating metrics.

Numerous metrics initiatives have failed because of a lack of awareness of this guiding principle. Sears proposed a metric for the design of human-computer interfaces called layout appropriateness [70]. The goal of the metric is to make it easier to arrange graphical user interface (GUI) components in a way that makes it easiest for users to interact with the software. One of the rare metric constructions for comprehending human-computer communications stands out thanks to Sears' work.

Based on the corresponding requirement specifications, researchers propose metrics set for evaluating the analysis model quality: accuracy, completeness, verifiability, understandability, external/internal consistency, concision, achievability, traceability, reusability, and precision. Several of such characteristics are typically regarded as highly qualitative. However, quantitative metrics are established for each by the authors. For instance, the proportion of several necessities for which all investigators had the same understanding to the sum of necessities is what is meant by the term "specificity," or lack of ambiguity.

In his article "Successfully Applying Software Metrics", researchers presents a set of tenets that he has learned from implementing metrics programs in a large organization [68]. He emphasizes four primary areas of focus that significantly influence the resultant of the entire metrics effort: progress monitoring as well as project estimation, evaluating work products, enhancing processes through failure assessment, as well as testing best procedures. In last, Grady offers the project managers involved in a metrics initiative the three recommendations listed below.

- At an early stage of your project, define your measures of success and monitor your progress toward them.
- Users can use trends in defect data to decide when to launch the product.
- Measuring complications can help you make better design choices as well as make the project easier to keep up.

The outcomes of computing software procedure upgradation initiatives in Siemens software development organizations are reported by Paulish and Carleton in 1994 [71]. The suggestions made by the researchers involve:

- Application of the Capability Maturity Model
- Performing analysis to initiate programs for software procedure development
- Choosing some procedures for improving the process and diligently implementing them

- Giving the method's implementation equal or greater attention than the method itself.
- Recognizing that process improvement methods vary in terms of how easy they are to introduce and use.

A metrics set for OO system design as well as development projects is provided by Lorenz and Kidd [72]. The authors introduce metrics to gain a deeper comprehension of and control over the development process, beginning with fundamental ideas like inheritance and class size. Specialization index, number of operations combined by a subclass, number of operations overridden by a subclass, and class size with other metrics are some examples. Experimental outcomes from schemes using Smalltalk, C++, and other programming languages support some metrics.

In two related papers, Chidamber and Keremer proposed one of the most widely cited sets of object-oriented metrics [73-74]. The six class-based design metrics with descriptive names that make up the set are now known as the CK metrics suite, weighted methods for each class, the number of children, the depth of the inheritance tree, the coupling among object classes, the response for a class, and the absence of method coherence. "This set of six metrics is depicted as the primary empirically authenticated suggestion for formal metrics for OOD," the authors of the latter paper assert after providing an analytical evaluation of all of the metrics. Additionally, the paper discusses a number of industrial software development uses for these metrics.

The experimental however argumentative problem of utilizing metrics to handle software projects is addressed by Weller [75]. The advantages that expediency metrics can bring to each of the three levels of project management are highlighted. The author comes to the conclusion that defect data can play a crucial role in enhancing project planning. However, he points out that developers' reluctance to share such data with management is the biggest impediment to any defect data-based approach. This, in addition to other human features of metrics-based techniques, is a problem that software engineering will always face.

In paper Software Measurement, Fenton: Software metrics should adhere to fundamental measurement theory principles, according to A Necessary Scientific Basis. Additionally, he asserts that "...the exploration for common software complication measures is destined to failure" as well as provides extensive analysis to support this assertion. A "Unifying Framework for Software Measurement" is proposed as a result of a review of the measurement theory principles that are carefully related to software measurement. Additionally, Fenton emphasizes the necessity of

validating software measures. The author observes that measurement theory has been the foundation for the most promising software metrics formulations.

Studies on software metrics typically overlook post-delivery issues. The loosely defined "maintenance" area is rarely the subject of systematic investigation. The IEEE proposed software maturity index (SMI)—IEEE, 1994—depicts on a software product's level of stability as it is sustained as well as improved via continuous post-production releases. This is a notable exception. The formula is  $SMI = [MT (Fa + Fc + Fd)]/MT$ , denoting the module number in the present release, the module numbers in the existing statement that have been altered, the number of modules in the present statement that have been integrated, as well as the module number from the previous statement that were erased in the existing statement, respectively [76]. The product begins to stabilize as SMI nears 1.0. Despite the fact that issues with maintenance, such as user ignorance, environmental failures, etc., can occur regardless of the modules added or modified, The SMI is, in fact, a useful abstraction for enumerating post-delivery difficulties in big software models. The significance of metrics in object-oriented testing is emphasized by Binder [77].

In point of fact, software testing, due to the simply quantifiable inputs it provides (number of units being experimented, attempts in person hours, etc.) and outputs (such as the number of faults and unit defects) is the progressing action that can be measured the most easily. Coupling/cohesion are the impactful concepts that describe few inherent properties of component communication. They can be compared to the yin and yang of software framework, opposing yet complementing forces that impact collaboration as well as constituent framework. Cohesion among software components has been extensively studied. They provide a collection of metrics that are defined in terms of the terms "stickiness," "glue tokens," "superglue tokens," and "data slice." Impactful operational cohesion and weak operational adhesiveness/cohesion—the relative amount to which tokens bind data slices combinably—are the metrics developed by the authors. The values of all of the cohesion measures are between 0 and 1. The researchers propose a metric for module coupling that includes global, environmental, as well as control/data flow coupling. Some proportionality constants whose values are determined by experimental verification are utilized in the module coupling indicator.

The Goal-Question-Metric method (GQM) has been adapted for software development by researchers. The researchers state, "GQM method is depending on the consideration that a firm must primarily declare the objectives for itself and its projects, then it must track such objectives to the information envisioned to explain those objectives functionally, and ultimately it must offer an

architecture for data interpretation against the stated objectives." There are three levels to this measurement model: operational level (QUESTION), conceptual level (GOAL), and quantitative level (METRIC) [78-79]. A "concept for interpreting/defining the measurable/operational software" is ultimately the goal of the strategy. It can be used on its own or, even better, as part of a broader strategy for improving software quality. "Outcomes of existing research works show that approaches tend to be small, both in as logical complexity as well as statement number, proposing that associated architecture of a model may be more significant with respect to the perspective of a particular modules," the researchers state, highlighting the concern with class as the dominant entity of interest in OO aspects. The researcher identifies three straightforward metrics that evaluate the features of the approaches: number of constraints, complexity and average size, required for each task.

The researcher investigates the unique place that object-oriented metrics hold in software metrics research. He identifies five distinctions between OO metrics [80]:

- Object abstraction techniques
- Inheritance
- Information hiding
- Encapsulation
- Localization

The author asserts in the article's opening section: "for a specific procedure, individual or product, six well-chosen metrics observes to be an experimental upper limit, that is additional metrics do not generally offer a substantial investment/return," states that software engineering metrics are rarely useful on their own.

The Team Software Process (TSP) as well as Personal Software Process (PSP) developed by Humphrey have gained widespread industry acceptance as efficient methods for increasing software development teams' and practitioners' productivity [79].

Humphrey shows how measurements can help people understand and use their own skills and expertise in a paper named as "Using a Defined and Measured Personal Software Process." Metrics can help a lot in this regard because continuous monitoring of the development process is a key component of Humphrey's methods.

The study's researchers introduce functional methods for measuring software process. Their strategy is primarily based on process measurement-specific function point analysis. The arguments are certainly strengthened by the chapter that discusses examples of successful applications of these strategies.

Scott A. Whitemire's *Item Situated Plan Estimation* is a fundamental work in the investigation of item arranged measurements [82]. Whitemire treats patients with extreme rigor: establishing the hypothetical bases, contextualizing measurement, as well as taking design features via his metrics. Complexity, size, sufficiency, coupling, cohesion, completeness, similarity, primitiveness, and volatility are all covered by Whitemire's metrics. Motives and ancestors, empirical perspectives, formal properties, empirical relational structures, potential measures, and so forth, within each area are talked about. The author offers a novel perspective on numerous software measurement issues. The development of a solid scientific architecture for comprehending as well as evaluating till date the most time-consuming perspective of software progress—architecture—is Whitemire's most significant contribution [82]. The book's heavy emphasis on rigor also has a disadvantage. Practitioners rarely have the time to fully comprehend and apply Whitemire's findings because of the arduous nature of industrial software development.

## **2.11 Metrics in the New Millennium**

Demeyer and et al. "... suggest a heuristics set for recognizing refactorings by putting OO metrics, lightweight to consecutive software system forms" [82]. The following assumptions are made by the authors regarding the consequences of specific code structural changes:

- **Method Size** – A reduction in method size becomes the sign of method split.
- **Class Size** – A shift in functionality to sibling classes (i.e., incorporating object configuration) is indicated by a decrease in class size. Additionally, it is an indicator of the redistribution of instance methods/variables inside the hierarchy, also known as optimizing the class hierarchy.
- **Inheritance** – When a class hierarchy is optimized, a change in the inheritance of classes is a sign.

The authors demonstrate significant relationship among design drift as well as refactoring and how metrics can assist in recognizing and comprehending them, despite the fact that such considerations are not without controversy—for instance, an approach may reduce the size because of the introduction of an intellectual programming, which is not unavoidably indicative of method

split. In his extensive book *Software Engineering*, Pressman delves deeply into the field of software metrics:

Pressman, *A Practitioner's Approach* is the textbook of choice for many graduate courses [83]. Pressman distinguishes between project management and process compliance metrics and so-called technical metrics, which aim to document the software product's development and behavior. Additionally, object-oriented system metrics are the subject of an entire chapter in the book. The inherent difficulties of iterative software development are highlighted by Sotirovski [84]: "... If the iterations are too small, iterating itself could consume more energy than designing the system." If it's too big, we might put in too much attempt before discovering the incorrect process to go. The investigator emphasizes the significance of heuristics in iteration monitoring/planning as a means of overcoming this obstacle. Heuristics frequently reflect the wisdom gained from successful metric efforts. Heuristics, on the other hand, are frequently essential for expediting software design and implementation in the absence of physical laws.

Lanza takes an interesting as well as unconventional method to a metrics-based understanding of software evolution [85]. The investigator suggests an Evolution Matrix that "... shows the evolution of a software system's classes." The software version is represented by each column of the matrix, and the various versions of the same class are represented by each row. Classes are broken down into groups with odd names based on this idea: Stagnant, Dayfly, Pulsar, Persistent, White Dwarf, Red Giant, and Supernova. Lanza identified distinct classes and phases in the evolution of a system based on data from case studies. Even though the method has some drawbacks, the paper still offers a novel perspective on how software systems change.

Software engineering research continues to focus on the importance of comprehending as well as justifying the impacts of alteration on enterprise software systems. It is fascinating to observe how Kabaili et al. [86] have attempted to understand cohesion as an indicator of OOSs' changeability. Empirical data from C++ projects has been used by the authors to support their assertions, which seek to establish a correlation between changeability and cohesion.

However, the authors conclude that coupling in comparison to cohesion appears to be a superior changeability indicator based on their research. This study employs a novel method to examine the possibility that design features may expose more information than was firstly intended. Mens and Demeyer (2001) emphasize the significance of retrospective/predictive



assessment in software evolution research in their research work about Future Trends in Software Evolution Metrics [87].

Despite the fact that some of the following areas have already been thoroughly investigated, they identify them as promising areas for future metrics research:

- Language independence
- Process issues
- Measuring software quality
- Data gathering
- Detecting and understanding different types of evolution
- Long term evolution
- Realistic case studies and empirical validation
- Scalability issues

Ramil and Lehman (2001) investigate the significance of measuring software evolution processes and products over the long term [88]. The Feedback, Evolution, and Software Technology (FEAST) program's empirical data are used in this example. The application of a sequential statistical test (CUSUM) to a collection of eight development event metrics is demonstrated in the example. The investigators emphasize the importance of having a precise definition of metrics because even minor definitional differences can result in an excessively wide range of measured values.

Given the business requirements that enterprise software must fulfill first, Rifkin provides a perspective on the reason why software metrics are so difficult to implement [89]. The attitudes toward measurements of four distinct domains of software development are examined and compared: The nonprofit sector, a Wall Street brokerage house, a civilian government agency, and a contractor for computer services. The author concludes that "We require to create an entire novel measures set for all those consumer-intimate as well as product-innovative firms that have prevented measurement thus far" and advocates a measurement strategy that is appropriate for each type of business.

It appears to be a deceptive reference to the classic article in Brooks' classic book *The Mythical Man-Month*, Fergus writes in his book *How to Run Successful Projects III – The Silver Bullet: In Essays on Software Engineering*, measurement methods can have a significant impact on project outcomes [89]. Particularly insightful is his probability of success indicator (PSI) metric.

An organization's software measurement initiatives typically focus on the tangible, such as developer productivity and lines of code. Buglione and Abran (2001) investigate the measurement of organizational creativity and innovation [90]. The authors examine, using the framework of popular software procedure development systems like P-CMM as well as CMMI, how creativity as well as innovation can be measured in both process and people aspects.

## **2.12 Crosscutting Concerns**

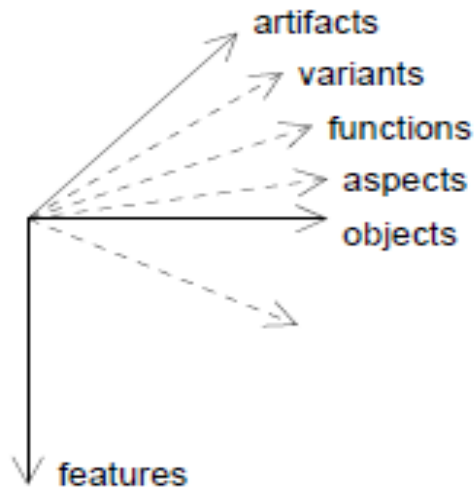
The programming paradigm known as aspect-oriented programming (AOP) aims to modularize issues that span multiple domains. A structural connection exists between the representations of two concerns, which is known as crosscutting. To put it another way, one concern's representation intersects with another concern's representation. Another structural association to block as well as hierarchical construction is crosscutting. It is not defined among concerns; rather, it is defined among their illustrations, which are the modules that execute the concerns. We have already discussed a cross-cutting issue in our FOP remarks: Collaborations cross module boundaries established by classes to extend a program in multiple locations. Collaborations are modularized by feature modules, which then implement features. AOP generally takes into account cross-cutting issues without putting a special emphasis on collaborations or feature modularity.

Crosscutting appears to be caused by a limitation known as the tyranny of the dominant disintegration, which is present in both conventional languages as well as modularization procedures: A coding can only be modularized in one direction (along one dimension) at a time, and the numerous different forms of issues that do not coordinate with that modularization result in code that is dispersed, tangled, and replicated. The various dimensions of concern separation, such as along the object or feature dimensions, are depicted in Figure 2.9.

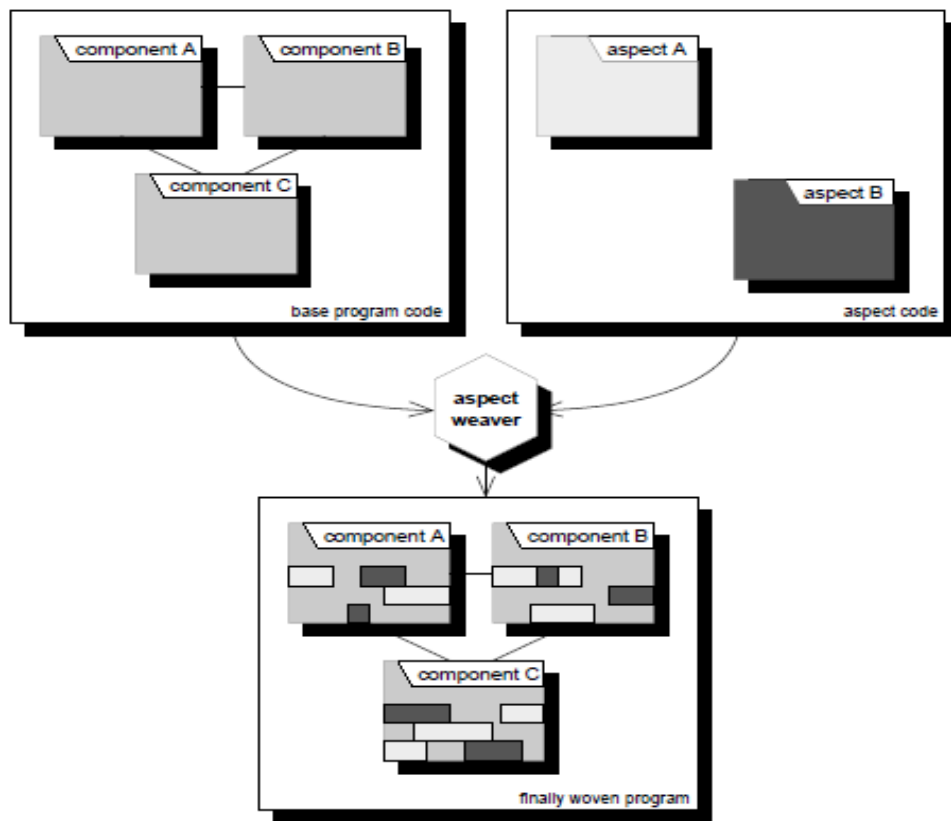
A concern implementation that is dispersed across a large number of other concerns implementations is referred to as code scattering. code tangling is the integration of multiple concerns' implementations within a module. Both violate the concept of data concealment and reduce modularity. In order to display the colors appropriately, the print approaches of Node and Edge are upgraded further [91-92]. Color's implementation is broken up into three classes (Color, Node, and Edge), as well as two methods in each of these classes change. In addition, it is intertwined with the feature Display, which is used to display the graph structure and is distributed across Edge, Node and Graph [93].

Code tangling as well as scattering make a coding harder to understand. When handling with tangled code that highlights many concerns the coder becomes disoriented. Programmers are forced to consider a problem at multiple points in a program when there is scattered code. Because the concerns become coupled and tangled/scattered code reduces customizability, maintainability and reusability overall. In other words, their implementation goes against the principle of concern separation. Code replication that typically happens when a concern associates with several concerns as well as the entire associations are executed likewise, is another negative effect of crosscutting [94-95]. For instance, the code for handling as well as varying colors that is duplicated in the classes edge as well as Node is the result of the execution of our feature Color. Code replication has been identified as a serious issue: In addition to the drawback of repeatedly implementing the same functionality, code replication makes it harder to maintain software and increases the likelihood of errors brought on by copying and pasting code fragments.

AOP addresses cross-cutting concerns' problems: concerns that are well-suited to modularization by means of a programming language's provided decomposition mechanisms (also known as these mechanisms are utilized in the implementation of host programming language) [96]. As so-called aspects, all other concerns that intersect with the execution of other concerns are developed. A type of module known as an aspect encapsulates the execution of a broad concern. It eliminates code tangling/scattering by allowing code connected with a single crosscutting concern to be contained in a single module. Additionally, aspects can prevent code replication by affecting many other concerns with a single piece of code. At join points that have been predetermined, an aspect weaver combines the various program features with the remaining code elements. Aspect weaving is the term for this procedure. Activities in the dynamic program implementation, such as a call to a method in the control flow of another method, or syntactical program elements like a class declaration are examples of join points [97].



**Figure 2.3 - Dimensions of Separation of Concerns**



**Figure2.4 - Aspect Weaving**

The traditional aspect, which is frequently represented to as a modularization concept, breaks the principle of information hiding: Even though the aspect itself has an interface, it directly influences other modules without using a line. Independent module development and modification is prohibited by this. On the other hand, it has been contended that conventional modularization

concepts do not adequately address cross-cutting issues. As a result, aspects seem like a sensible alternative. There are a number of initiatives aimed at uncovering previously hidden information in AOP [98].

The idea of an aspect expands the idea of a class in the majority of AOP languages. Aspects may also consist inter-type, advice and pointcuts declarations, in addition to the organizational fundamentals recognized from OOP, such as fields/approaches.

**Pointcuts:** A pointcut is an expression (quantification) that estimates whether a provided join point matches. It is a declarative requirement of the join points into which an aspect will be woven.

**Advice:** The guidelines that are assumed to be carried out at a join points set are encapsulated in an advice, which is a method-like component of an aspect. The set of advised join points is defined by pointcuts that are bound to advice pieces.

**Declarations between types:** From within an aspect, interfaces, fields and methods can be integrated to current classes with an inter-type declaration.

This chapter explains the reaserch work already get done in past and also explains the future scope suggested by different researchers.

## CHAPTER3

### METHODOLOGY TO DESIGN EMPLOYEE MANAGEMENT SYSTEM

The EMS is a distributed application designed to keep track of employee information across all businesses. It keeps track of their employees' personal information as well as information about the payroll system that enables them to generate payslips. The application used during this work is a collection of Java-based applications. It is easy to use and can be used by anyone, even if they have never used a simple employees system before. It is easy to use and simply requests that the client follow bit by bit tasks by giving him few choices. It can carry out numerous company operations and is quick.

AOP and OOP on the front end and Microsoft SQL Server on the back end were used to create this software package. The software is extremely simple to use. Different modules, like Employee Details, are included in the package. Further, the software in this version supports multiple users. As this package organizes and simplifies team management, an employee management system becomes a beneficial tool for managers, HR professionals, as well as business owners alike. These systems can track performance and time off, payroll, onboarding, and other HR functions, among other things. Finding a model that not only functions well for your team but also saves you time, effort, and money is the key. One can make better use of your time by using a web-based employee management system, which also cuts down on the time your team spends doing mundane administrative tasks [99].

An application known as an EMS enables users to create and store Employee Records. The application likewise gives offices of a finance framework which empowers client to produce Pay slips as well. This application is useful to branch of the association which keeps up with information of representatives connected with an association.

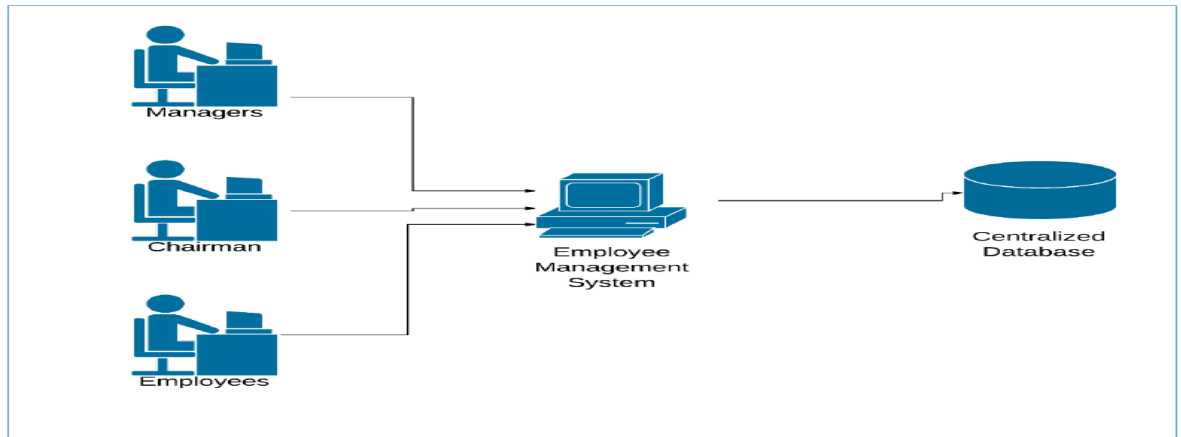
Python is a language that works on any platform. Applications it creates can be used on a distributed network or on a single machine. Furthermore, Python-based applications can be extended to Internet-based applications. User can authenticate the entire necessary data about your customers, projects and team, from a solo dashboard if you have an employee management system in place. Every good employee management system should provide this centralized insight. Additionally, user should seek a system that offers:

- Increased employee engagement
- Data security

- Streamlined admin tasks and optimized workload

New metrics are supported in two main ways: aspect construction and metric definition.

- The definition of the entities to be measured (such as operations and fields) is the first part of the metric definition. and
- how they are combined (for instance, in aspects or classes).



**Figure 3.1- High Level Architecture of EMS**

This work must write an aspect that encapsulates the measurement procedure for the new metric defined in the first step during the aspect building step. Therefore, we must:

- Define a set of pointcuts that the measurement process needs to select or exclude certain execution points (also known as join points).
- Write the instructions for analyzing the selected joint points and collecting the metric data.
- Write a collection of ancillary classes to help support the measurement, such as to store data temporarily.

One of the most common management issues that businesses face in today's competitive business environment is, how to effectively manage employees, given that each person and employee is unique. Employees with impressive resumes and excellent credentials can definitely be hired by businesses. But effectively managing employees and dealing with management issues is just as important as hiring employees with the right experience and education to build a strong employee base that will be crucial to success in the future. In view of the abovementioned, this study will zero in on a few worker the executives issues, for example, the impacts of

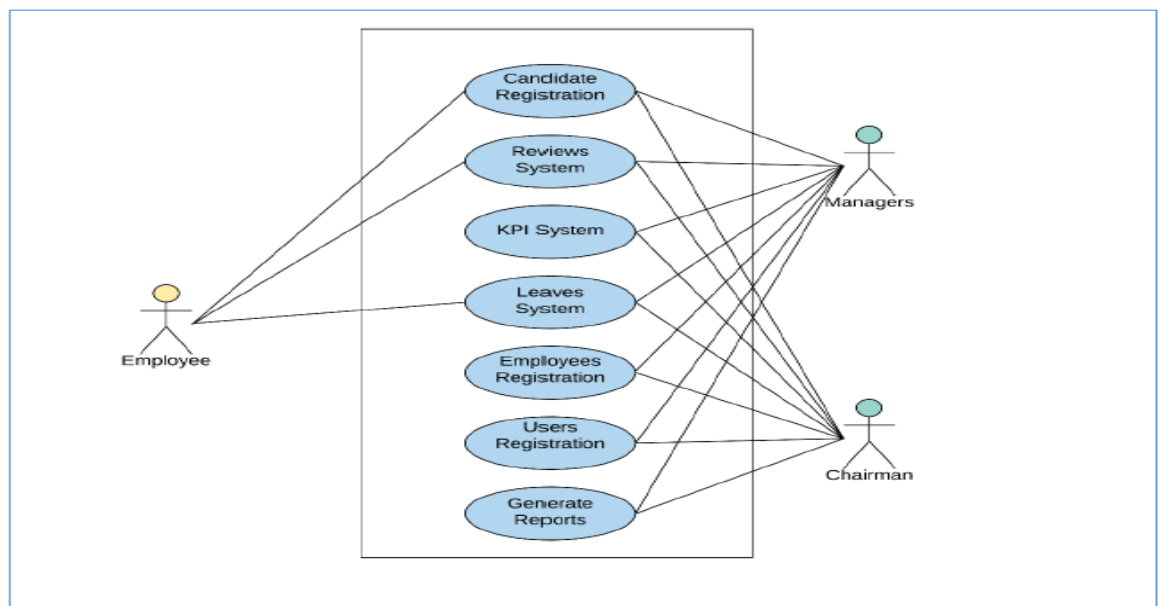
- Unfortunate administration

- b) Not persuading representatives really
- c) Not having the option to fittingly oversee struggle.

### 3.1 Planning

The software improvement procedure, from requirement analysis to maintenance/testing in accordance with the methodologies, that is carried out within a predetermined amount of time to achieve the desired outcomes is known as a software project. Planning was carried out in this project in an appropriate and precise manner. There are five fundamentals to the system:

- An employees database;
- Reduce time and cost
- Protect Fraudulent activity
- Online leave system
- System Manager for organization’s structure set-up;



**Figure 3.2- High Level usecaseDiagram of EMS**

The opportunities of this research work are as follows:

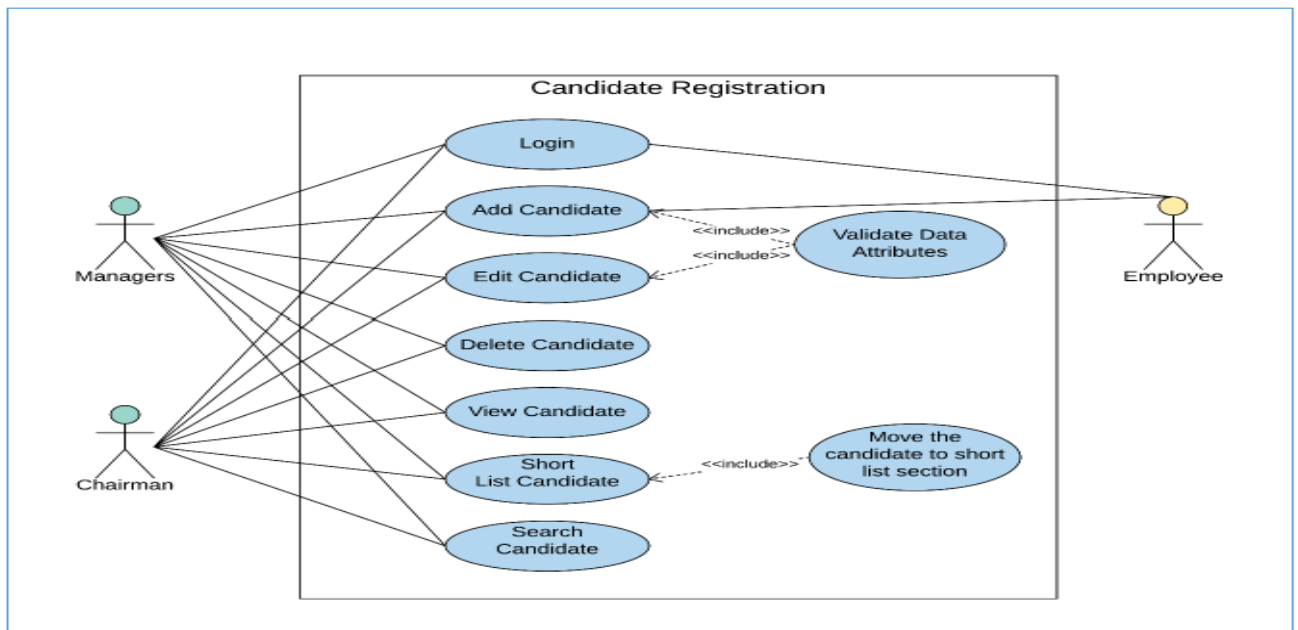
- a) Overall control and permission by Admin
- b) Online Reports
- c) Email notification
- d) Online leave module



- e) Company/registration user
- f) Requirement, Information and Company Setup.

The main advantages of an efficient EMS are:

- a) *Employee Engagement*: Employees feel more empowered and engaged when your company has a self-service portal.
- b) *Error-free Payroll*: With accurate attendance, working hours tracking, and legal compliance, the software automates payroll.
- c) *Access to Information*: The portal provides staff members with access to basic information without requiring them to contact human resources.
- d) *Remote Accessibility*: With this system, your employees can work from home and easily record their attendance.
- e) *Automate Data Entry*: The cloud-based framework permits you to computerize and digitize manual information passage saving time and exertion.
- f) *Easy to Customise*: EMS is an easy-to-customize system that lets you adapt to your organization's changing needs.



**Figure 3.3 - Candidate Registration Usecase Diagram**

The Candidate Registration module manages candidate registration based on the job title and has the ability to classify candidates into one of three statuses: Short Listed, Considerable, or Rejected.

Candidates can be registered by users at any level. However, just the clients with Administrator honors can waitlist them as per the sculptures referenced previously. The candidate registration use case diagram is shown in Figure 3.3.

This research work have various types of modules includes-

**1. Admin: The admin part mainly involves:**

- a) Dashboard: Administrators can quickly view every detail in this section, including the total number of registered employees, listed departments, leave types, unused leave, new leave requests, approved leave requests, and rejected leave requests.
- b) Department Administrators can add, update, and delete departments in this section.
- c) Leave Type The admin can manage leave types (add, update, and delete) in this section.
- d) Employee: The employee can be managed by admin in this section (add, update, delete).
- e) Salary: The admin can add, update, and delete salary in this section.
- f) Demand for Leave: The administrator can manage and update leave requests in this section.
- g) Reports: In this segment administrator, can see the number of representatives that have been enrolled in a specific period.
- h) Administrator can likewise refresh his profile, change the secret word and recuperate the secret word.

**2. User: An employee**

- a) Dashboard: It serves as an employee welcome page.
- b) My Account: Employees can view and update their profiles in this section.
- c) Leave: Employees can apply for leave and view their leave history in this section.
- d) History of Pay: Employees can examine their salary history in this section.
- e) Representative can likewise see their profile, change their secret word and recuperate their secret key.

### **3.2 Methodology**

EMS is a software for managing employees that can make your workforce happy and keep employees happy. Employees require a positive work culture and environment. A tool that improves employee database management, payroll, and other features is an employee management system. Employees' work experiences are enhanced as a result.

Employees are happier when they are given recognition for their efforts and dedication to the company. Representative administration frameworks can likewise deal with your worker

acknowledgment program. The staff is empowered by all of the features, such as salary slips that are filled with data. The downloadable reports likewise further develop HR-representative relations accordingly, further developing labor force maintenance and efficiency. In the methodology, we explain how this presented work will achieve its goals. In methodology, we can describe the knowledge and resources one will use to complete the presented work to demonstrate its viability. To develop a web-based employee management system for any business, the following steps are taken:

- a) To begin, a requirement analysis was carried out in accordance with any business organization's HR and administrative procedures. The "software requirements specifications (SRS)" document has been prepared.
- b) The entity relationship diagram (ERD) and data flow diagram (DFD) have been created. User interfaces and types were created for various operations, including an online registration form and an application status view for employees, after the database design was completed. The dynamic scripting language for the web-based software was then Hypertext Preprocessor (PHP). Cascading Style Sheets (CSS), HyperText Markup Language (HTML), and jQuery were utilized on the software's front end [100]. The database was created with MySQL [101]. Additionally, printable reports were generated with the help of the pdf library.
- c) After the software was made, it was tested thoroughly with real-world use case scenarios.

We are developing Employee Management System APIs where the HR or the company management can trigger those APIs and perform various onboarding and release tasks like hiring, training, salary calculation, payroll generation, resignation cases and re-onboarding of an employee. We are using below tech stack [102].

- a) AOP with the Spring-boot architecture
- b) GIT for the version control
- c) MySql DB to store the employee records.
- d) Maven

Software known as an employee management system enables your employees to give their all each day to help your business achieve its objectives. It directs and oversees the efforts of employees in the right direction. Additionally, it safeguards the personal and work-related information of your employees. When needed, this makes it simpler to store and access the data.

In the worker the executives framework, you can oversee administrator exercises in a more straightforward and faster manner. Company's bottom line is ultimately impacted by the work of its employees, who are an essential component. It is a crucial component of HR management. It also contributes to employee engagement, reduces costs, and improves productivity through performance management.

An intelligent module in our employee management system helps you stay organized. With the product, you can get every one of the information of your representatives readily available. The cloud-based nature of employee management software is an advantage. Giving you access any place you are at. You don't have to keep the decisions because you don't have enough information.

Assuming you are voyaging and you want the assistance of a portion of your workers, you don't have to stand by till you arrive at the workplace. The information about your employee can be accessed with just a few clicks using this EMS. They don't have to sift through the database, which also saves a lot of productive time. HR team can assist you in putting some strategic decisions for increased productivity into action during this time.

Organization can use the standard metrics for employee management with employee management software. Additionally, handler can easily customize or create their own metrics to meet the requirements of any company's staff management. The EMS provides their client with seamless assistance with employee performance management and has an intuitive user interface.

Organization can get a better look at the entire staff management through the software. The tools, such as an analytically driven metric system, are also provided by the employee management system. With the timesheet the board and time following programming, it is across the board asset and the right worker the executives situation for significant information and shrewd decision making for your business.

For an organization's HR team, managing payroll is one of the most challenging responsibilities. They need to oversee it, remembering numerous things like assessments, advantages of different organization approaches, and different allowances. This may appear to be doable to some extent, but because it is a demanding and time-consuming task, it may also be susceptible to errors.

To compute the finance accurately it is likewise vital to effectively approach the representative data set. This is where EMS, the best employee management system, comes in handy. With our cloud-based worker compensation the board programming you can likewise robotize exact finance. This will assist you in gaining employee trust. Employee management systems can also prevent legal issues.

This system's development process is comparable to that of web-based applications. Because it is impractical to develop the entire system at once, software is developed incrementally so that it can be reused [15]. A method known as incremental development involves developing a system in a series of versions, or increments, each of which adds functionality to the previous one.

*a) Systematic Review:*

Representative administration framework to be grown with the end goal that it is fit for stamping participation of every worker. Users' data needs to be safe and easy to access whenever needed. to be structured so that it can be used again. An important consideration in determining an employee's salary is how holidays are managed. Applications ought to be fit for giving compensation, complete working hours, extra time, present days toward the finish of month in a tick.

*b) Strategy:*

Step-by-step planning and a timetable are part of this section of development. It is necessary for the development process to run smoothly and on schedule. It includes planning the steps needed to carry out the project, achieve its goals, and use it in a way that doesn't cause problems in the future.

c) *Design Assessment:*

A step known as design analysis involves planning and evaluating each screen design to determine whether or not it can achieve the desired outcomes. When necessary, this step is repeated. If a new or updated feature is required, the process begins here. The most difficult aspect of this project was designing a user interface that was straightforward. It is a step-by-step process in which initial design is completed so that it can implement all necessary functionalities and later design can be made attractive.

d) *Building a GUI:*

The app's graphical user interface (UI) is developed using code after obtaining a clear picture of the screens and their design. Since this development replaced Flutter, a single language is used to design and implement backend processes, making this technology much simpler and more effective. The task is to put it into action and deal with any errors that occur, for which we must search the internet. This work involves the reading of official documents and other online sources for this section.

e) *The Design and Implementation of a Database:*

Databases and classes are designed in accordance with the functionality we want to provide when designing a backend. This work used Cloud Firestore for databases. This was the most difficult part for me because this work had to keep thinking about how data should be stored so that it can be easily accessed and not mixed up. There was a high risk of having data stored more than once because it would be used in multiple places. Another task was to store data in a way that kept it separate, and another was to figure out what and where the required fields were.

f) *Bringing Database and UI Together:*

Now comes the most crucial part, image, where our functions need to store and retrieve data. Since data needs to be updated, we need to remember how the data is used to ensure that nothing is lost. The data can vanish if handler make one mistake, but this only happens once while developing.

g) *How it's done:*

Execution is a stage where we would be fostering our functioning application. Here, the functionalities and data are put into action and used as needed. Here, the whole code is put into action and checked for errors. The majority of the development takes place here, and after this step, we get our finished product.

h) *Evaluation:*

This is the most frequently involved step; after completing each task, we must test. Additionally, once new functionalities are developed, we must recheck existing ones. We must ensure that the new thing does not affect the previous one. This work used the Android Studio-provided run emulator to test my application.

*i) Implement:*

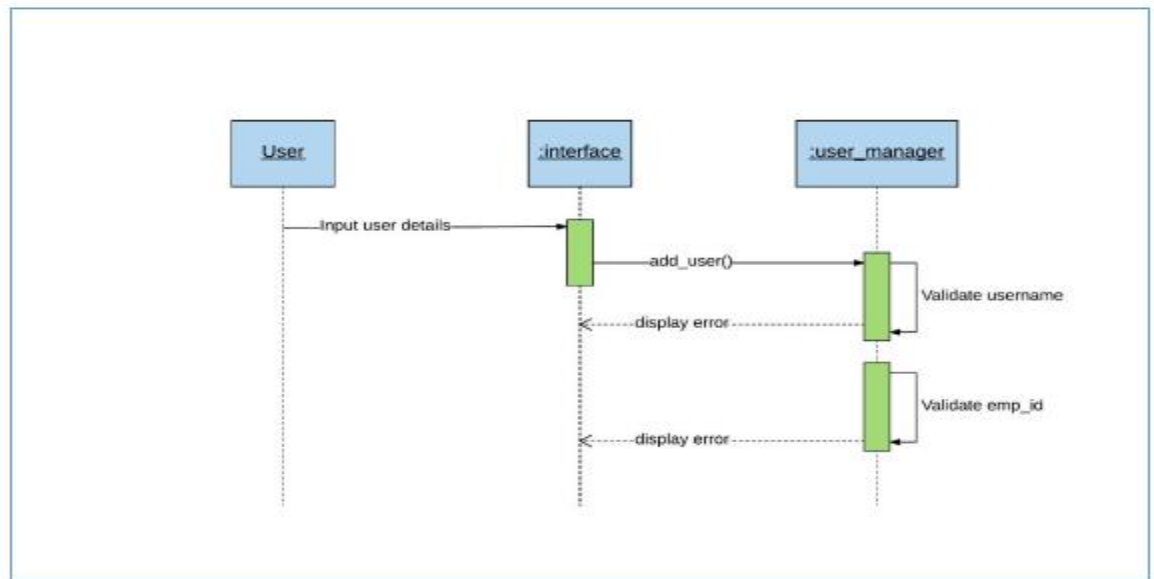
In the wake of getting the two applications created and passing their testing, the framework is fit to be executed in enterprises. It is presently fit to be utilized by society and further advancement will be proceeded with lifetime as new innovation and thoughts show up.

*j) Maintenance and Updating:*

Application update and maintenance is a lifelong process that will continue as we discover bugs and issues.

### 3.2.1 Spring AOP Capabilities and Goals

Pure Java is used to implement Spring AOP. There is no special compilation procedure required. Spring AOP is appropriate for utilization in application server or Servlet container because it does not require to regulate the class loader hierarchy [103].



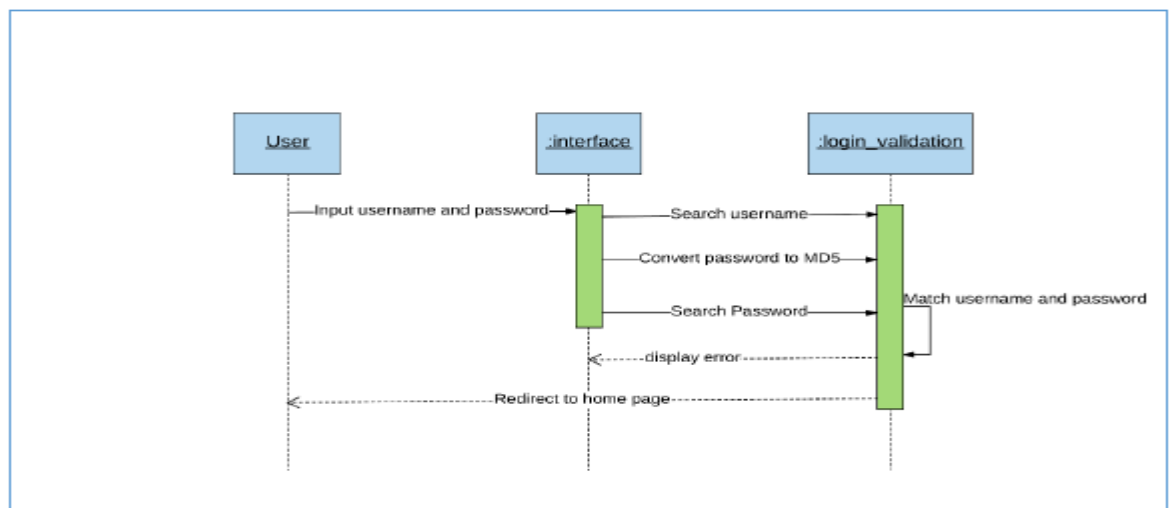
**Figure 3.4 - Login Flow Diagram of EMS**

Spring AOP currently just supports method implementation join points, which help Spring beans execute methods. Support for field interception could be integrated without disintegrating the core Spring AOP APIs, but it is not currently implemented. Consider using a language like AspectJ if you require to advise field authentication or upgrade join points. Spring AOP takes a different

approach to AOP than the majority of other AOP frameworks. Even though Spring AOP is quite capable, the objective is not to offer the major comprehensive AOP execution; Instead, the goal is to facilitate close integration among Spring IoC and AOP execution in order to assist in resolving common issues in enterprise applications [104].

As an illustration, the Spring IoC container and the AOP functionality of the Spring Framework are typically utilized together. Although this enables powerful "autoproxying" capabilities, aspects are constructed utilizing normal bean definition syntax: Compared to other AOP implementations, this is a significant distinction [105]. With Spring AOP, you can't do some things quickly or easily, like advise very fine-grained objects (like domain objects generally): In these situations, AspectJ is the best choice. Although, we have found that enterprise Java applications that are agreeable to AOP benefit greatly from Spring AOP's excellent solutions to the majority of issues [106].

Spring AOP will never attempt to contest with AspectJ in terms of offering an all-encompassing AOP explanation. Investigators consider that full-blown frameworks like AspectJ and proxy-based architectures such as Spring AOP are valuable and work together rather than against one another. With Spring 2.0's seamless integration of AspectJ with Spring AOP and IoC, a steady Spring-based application framework can accommodate all AOP applications [107]. The Spring AOP API and the AOP Alliance API are unaffected by this integration: Spring AOP is compatible with previous versions.



**Figure 3.5- Password Protected Login Page of EMS**

Pointcut reuse is made possible by Spring's pointcut model regardless of the advice type. Using the same pointcut, you can target different advice [108].



The organization Spring Framework The central pointcut interface is utilized to tailor advice to specific classes as well as approaches. The entire interface is displayed below:

```
publicinterfacePointcut {  
  
ClassFiltergetClassifier();  
  
MethodMatchergetMethodMatcher();  
  
}
```

Fine-grained arrangement processes, like conducting a "union" with another method matcher, are made possible by disintegrating the Pointcut interface into two sections.

The pointcut can be restricted to a particular set of target classes by using the ClassFilter interface. The matches() approach will match all target classes if it always returns true:

```
publicinterfaceClassFilter {  
  
booleanmatches(Class clazz);  
}
```

The MethodMatcher interface is generally more significant. The complete interface is depicted as:

```
publicinterfaceMethodMatcher {  
  
booleanmatches(Method m, Class targetClass);  
  
booleanisRuntime();  
  
booleanmatches(Method m, Class targetClass, Object[] args);  
}
```

Pointcut operations are supported by Spring: particularly, intersection and union.

- Union refers to the methods that match either pointcut.
- The methods that match both pointcuts are referred to as intersections.
- Typically, union is more beneficial.
- The *org.springframework.aop* support's static methods can be used to create pointcuts utilizing the ComposablePointcut class in the equivalent package or the Pointcuts

class [109]. However, using pointcut expressions from AspectJ typically proves to be simpler.

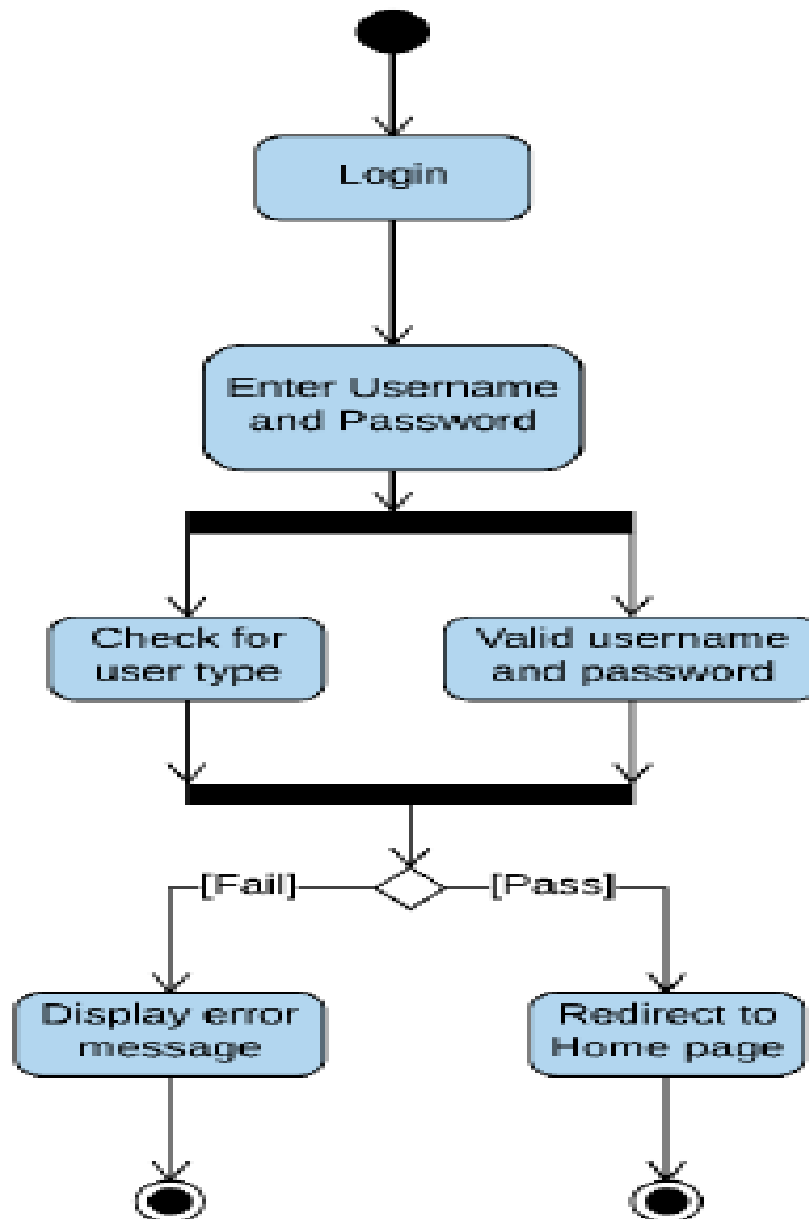
- *org.springframework.aop.aspectj* is Spring's most important pointcut type along with *AspectJExpressionPointcut* [110].

This pointcut parses an AspectJ pointcut expression string with the help of a library provided by AspectJ.

The classes and structure of the project that we will use for the development are listed below.

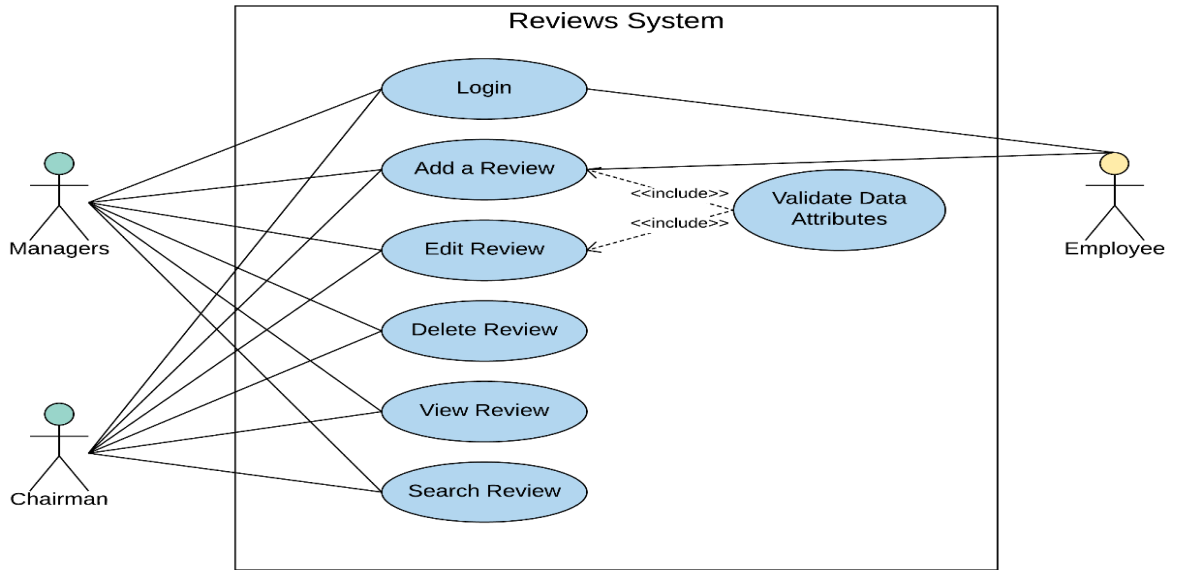
### **3.3 Reserach Structure**

Use the simplest option that can be utilized. Because the user don't have to integrate the AspectJ compiler and weaver into your progress as well as construct procedures, Spring AOP is easier to use than full AspectJ [111]. Spring AOP is the best option if you only need to recommend on how to run operations on Spring beans. User will need to use AspectJ if you requisite to advise objects that are not handled by the Spring container, like domain objects generally, or if the user want to advise join points other than simple method implementations (like set join points or field get, etc.)



**Figure 3.6 - User Login Activity Representation of EMS**

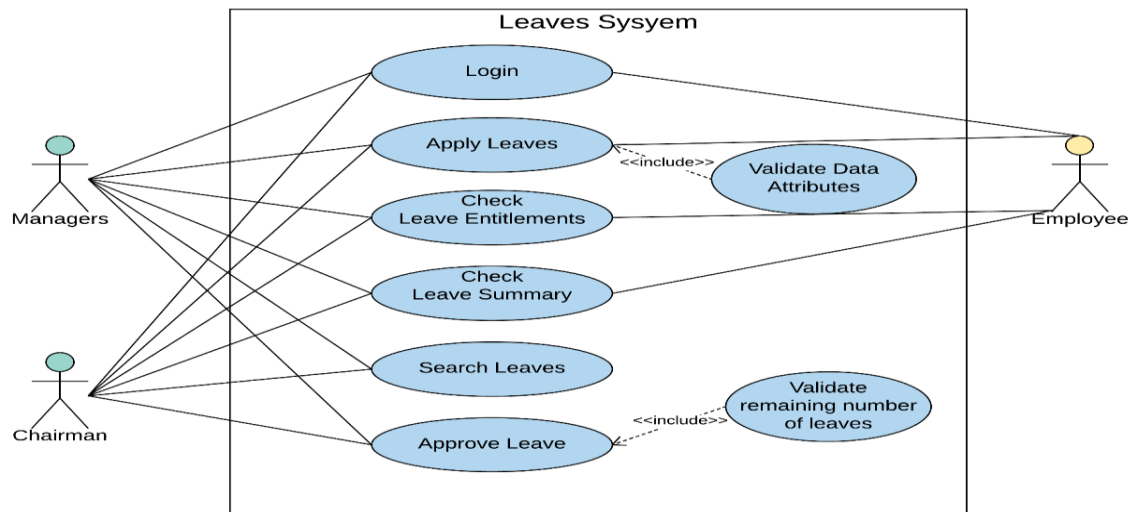
User can use either the @AspectJ annotation style or the AspectJ language syntax when using AspectJ (also named as the "code style") [112]. It goes without saying that if you don't use Java 5 or later, the selection is already made for you: use the code style. The AspectJ language syntax is chosen if aspects play a significant role in the framework and user are capable to utilization the AspectJ Development Tools (AJDT) plugin for Eclipse: Because it was designed specifically for writing purposes, it is simpler and cleaner [113].



**Figure 3.7-A Review System of EMS**

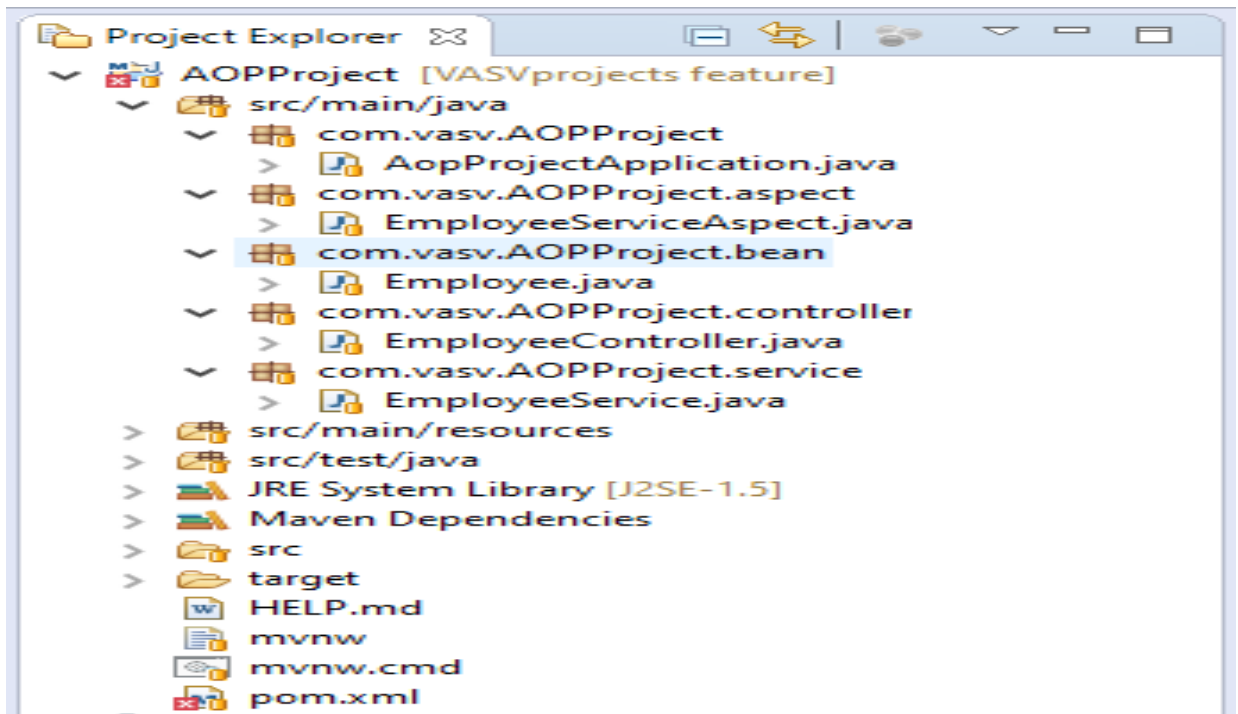
The review system module handles employee reviews, which each employee can give to other employees. A review can only be edited or deleted by Admin users. The Review system's use case diagram can be seen in figure 3.7.

User might want to think about utilizing the @AspectJ style, using aneven Java compilation in your IDE, as well as combining an aspect weaving stage to your construct script if you don't use Eclipse or only have a few aspects that don't play a big role in the application of Employee Management System.



**Figure 3.8- Leave System of EMS**

It becomes important to calculate the number of leaves taken by the employee by both AOP and OOP systems in order to maintain the efficient environment in an organization. At the end of the month, it will help in the salary calculation and balance leave calculation. The annual, medical, and casual leaves that employees take are handled by the leave module. Each worker should get the endorsement prior to withdrawing. After approving a leave, the appropriate message will be sent to the Admin user, who is the only person who can do so. After reaching their leave quota, no employee may take any leave. Graph in figure 3.8 portrays the utilization case chart of Leave module.



**Figure 3.9- Project Structure of Employee Management System**

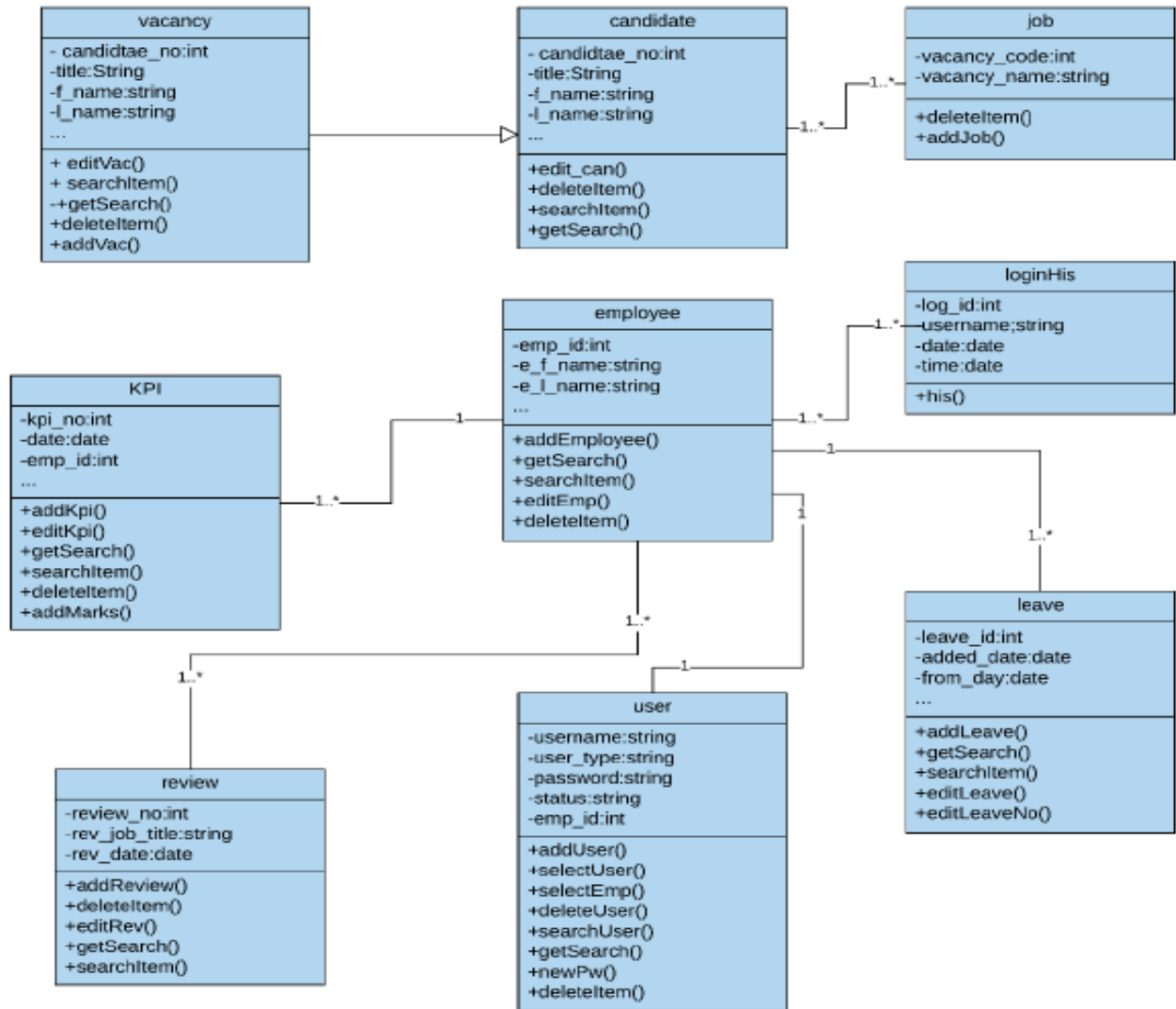
### 3.4 Classes

#### 3.4.1 Application Class

@AspectJ or XML style are the users preferred options if they have decided to use Spring AOP. Clearly, the XML style is the best option if they are not using Java 5+; There are numerous tradeoffs to consider for Java 5 projects [114].

Existing Spring users will be most acquainted with the XML style. It is supported by genuine POJOs and can be utilized with any JDK level (although using named pointcuts within pointcut expressions still requires Java 5+). XML can be a better selection for configuring

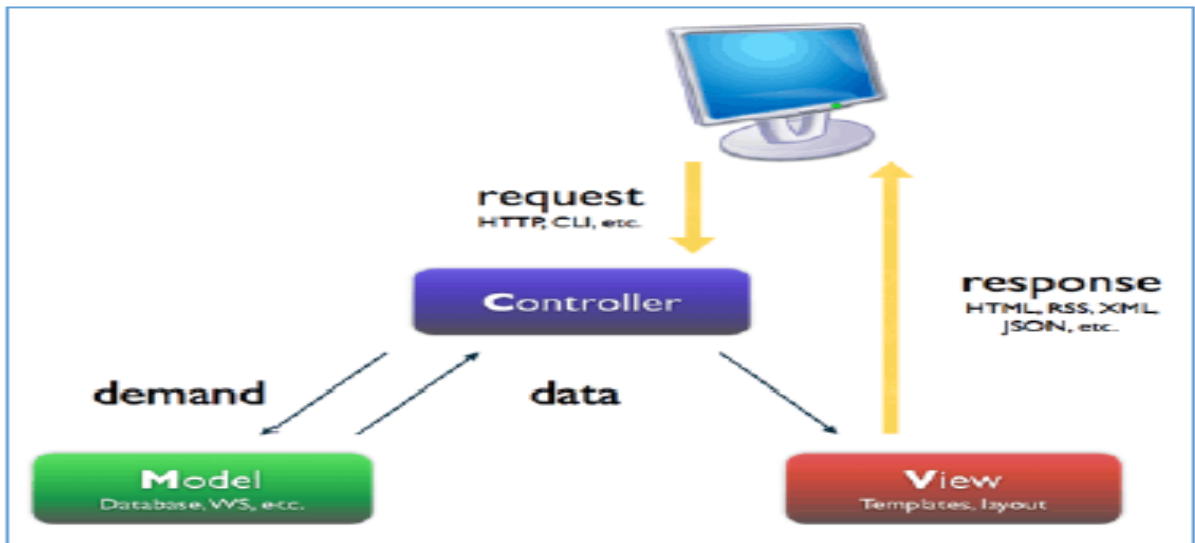
enterprise services using AOP (a better examination is to determine whether you taken into account the pointcut expression to be a portion of the structure that user might want to alter on their own). Using the XML format, it might be easier to see from your configuration which parts of the system are there [115].



**Figure 3.10- The Entire Class Diagram of the Proposed EMS**

There are two drawbacks to the XML style. First, the execution of the need it outlines is not completely encapsulated in one location. Any piece of understanding in a system ought to have a single, unambiguous, authoritative representation, according to the DRY principle [116]. The XML in the structure file and the backing bean class declaration comprise the understanding of how a prerequisite is executed when using the XML style. This data is contained within a single module,

the aspect, when employing the @AspectJ style. Second, compared to the @AspectJ style, the XML style is slightly less expressive: Named pointcuts declared in XML cannot be combined, and only the "singleton" aspect instantiation model can be used.



**Figure 3.11-System Architecture of the Proposed EMS**

The system architecture of an EMS by using AOP and OOP is shown in Figure 3.11, while its use in application class is demonstrated in Figure 3.12.

```
Employee.java  AopProjectApplication.java
1 package com.vasv.AOPProject;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableAspectJAutoProxy(proxyTargetClass=true)
9 public class AopProjectApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(AopProjectApplication.class, args);
13     }
14
15 }
```

**Figure 3.12- Application Class of Employee Management System**

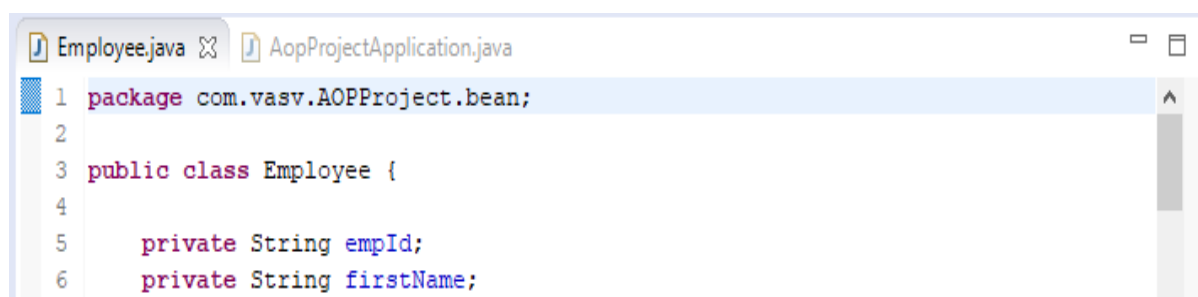
### 3.4.2 Bean Class

If users are using STS, you can make a "Spring Bean Configuration File" as well as select the AOP representation namespace; however, if user are utilizing another IDE, you can just integrate it to the spring bean structure file. The configuration file for this project bean is as shown below [116] spring.xml:

The Spring Framework documentation offers the subsequent definition of beans:

Beans are the Spring name for the objects that make up your application's backbone and are handled by the Spring IoC container. A Spring IoC container is responsible for managing, assembling, and instantiating an object known as a bean.

This definition is brief and to the point, but it leaves out an essential component: the container for Spring IoC.

A screenshot of an IDE window showing two tabs: 'Employee.java' and 'AopProjectApplication.java'. The 'Employee.java' tab is active, displaying the following code:

```
1 package com.vasv.AOPProject.bean;
2
3 public class Employee {
4
5     private String empId;
6     private String firstName;
```

**Figure 3.13- Bean Class of Employee Management System**

### 3.4.3 Controller Class

The controller class in Spring Boot is in charge of responding to incoming REST API requests, creating a model, and recurring the opinion that will be showed as a response.

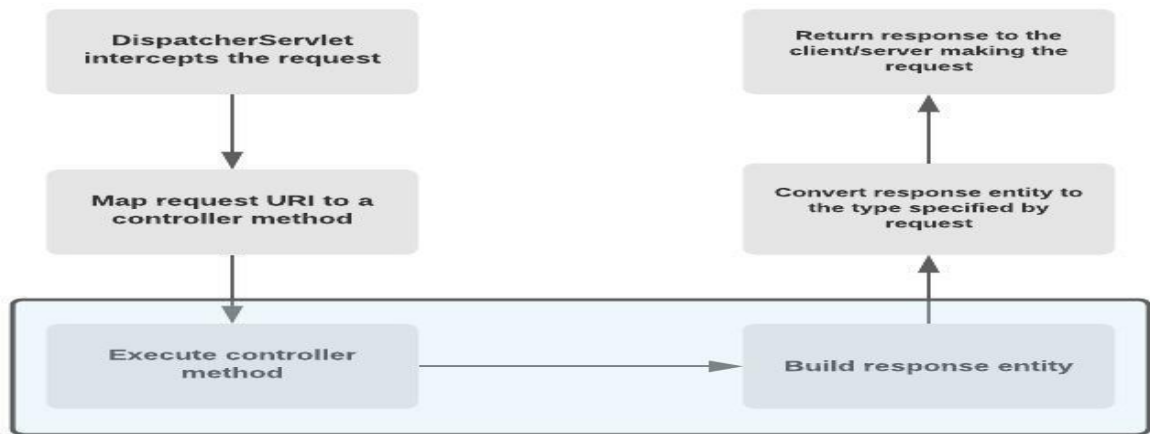
The @Controller or @RestController annotations are used to annotate Spring's controller classes. To enable Spring to recognize a controller class as a RESTful service at runtime, these mark it as a request handler [117].

The @Controller and @RestController annotations' definitions, application scenarios, and distinctions will be discussed in this tutorial.

Check out our comprehensive How to Build a Spring Boot REST API guide if you're new to Spring Boot.

Before defining the two annotations, we'll quickly go over the process by which Spring Boot processes and returns a response to REST API requests:





**Figure 3.14 -Working of Controller Class in an Employee Management System**

The DispatcherServlet is the first to receive the request. It is in charge of processing any incoming URI requests and mapping them to controller method-based handlers. The resource is then processed into a JSON or XML response following the execution of the controller method [118].

The two processes encapsulated in the rectangle in the preceding diagram represent the actual processes implemented by a developer. The DispatcherServlet and the rest are carried out by Spring services that are running in the background [119].

```

Employee.java  AopProjectApplication.java  EmployeeController.java
1 package com.vasv.AOPProject.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
11
12 @RestController
13 public class EmployeeController {
14
15     @Autowired
16     private EmployeeService employeeService;
17
18     @RequestMapping(value = "/add/employee", method = RequestMethod.GET)
  
```

**Figure 3.15 - Controller Class of Employee Management System**

### 3.4.4 Aspect Class:

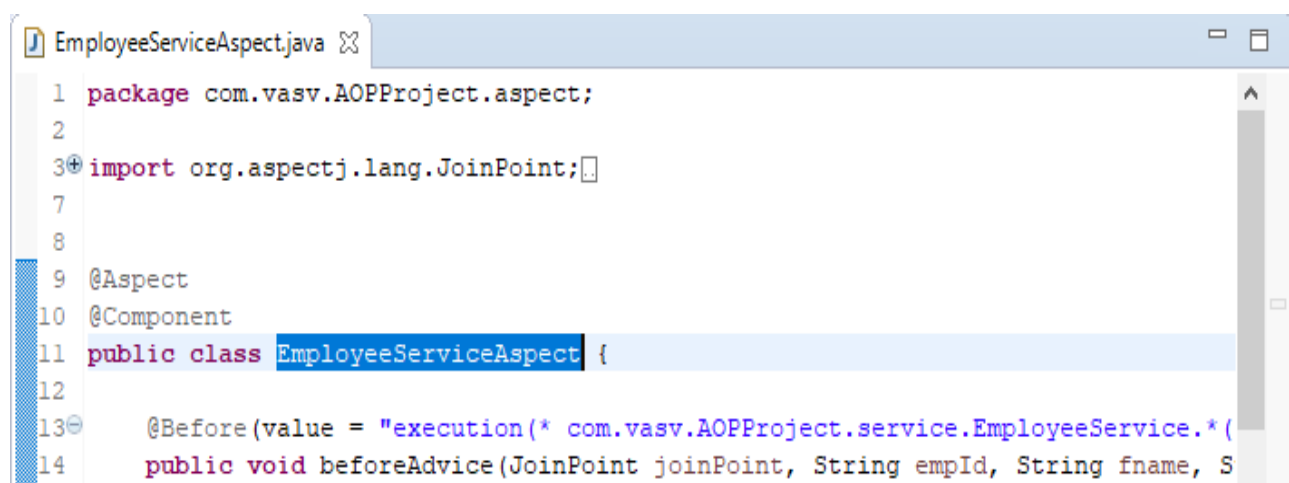
An aspect is a concern's modularization across multiple classes [120]. One example of this kind of interconnected concern could be unified logging.

Let's examine the definition of a straightforward aspect:

```
public class AdderAfterReturnAspect {  
    private Logger logger = LoggerFactory.getLogger(this.getClass());  
    public void afterReturn(Object returnValue) throws Throwable {  
        logger.info("value return was {}", returnValue);  
    }  
}
```

In the preceding instance, this work explained a straightforward Java class with a method called *afterReturn* that logs in one Object-type argument. It is important to keep in mind that even our *AdderAfterReturnAspect* class does not contain any Spring annotations [121].

Further research will see how to wire this Aspect to our Business Object in the following sections.



```
EmployeeServiceAspect.java  
1 package com.vasv.AOPProject.aspect;  
2  
3 import org.aspectj.lang.JoinPoint;  
7  
8  
9 @Aspect  
10 @Component  
11 public class EmployeeServiceAspect {  
12  
13     @Before(value = "execution(* com.vasv.AOPProject.service.EmployeeService.*(  
14         public void beforeAdvice(JoinPoint joinPoint, String empId, String fname, S
```

Figure 3.16- Aspect Class of Employee Management System

### 3.4.5 Service Class:

This class develops a service class to function with the Employee bean [121].  
EmployeeService.java code:

```
package com.journaldev.spring.service;
```

```
import com.journaldev.spring.model.Employee;
```

```

public class EmployeeService {

    private Employee employee;

    public Employee getEmployee(){
        return this.employee;
    }

    public void setEmployee(Employee e){
        this.employee=e;
    }
}

```

This work could have been configured as a Spring Component with the help of Spring annotations; however, in this project, we will use XML-based construction. The *EmployeeService* class is very common, as well as all it does is give us access to Employee beans.

```

EmployeeService.java
1 package com.vasv.AOPProject.service;
2
3 import org.springframework.stereotype.Service;
4
5
6
7 @Service
8 public class EmployeeService {
9
10     public Employee createEmployee( String empId, String fname, String sname)
11     {
12         Employee emp = new Employee();
13         emp.setEmpId(empId);
14         emp.setFirstName(fname);
15         emp.setSecondName(sname);

```

**Figure 3.17 - Service Class of Employee Management System**

The following steps must be taken in order to use Spring AOP in Spring beans:

- a) Like xmlns, declare the AOP namespace: "https://www.springframework.org/schema/aop" is the aop value [122].
- b) Add "aop": element aspectj-autoproxy to enable auto proxy at runtime support for Spring AspectJ

- c) Configure Aspect Classes in the same way as other Spring beans

As in this process, many aspects defined in the spring bean structure file; now it is time to investigate each aspect one at a time.

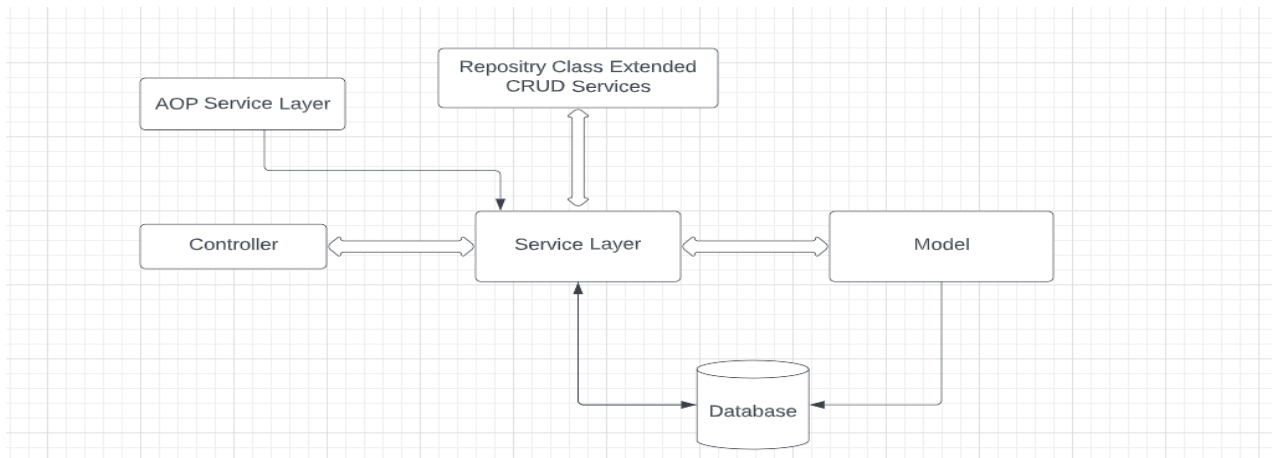
### 3.5 Architecture Diagram

The refactoring only addressed a few issues; As a result, not all of the issues that have been addressed in the constituents under deliberation have been repropotioned. The modules that were repropotioned for the system are those that implement the subsequent interconnected issues:

- a) Exceptional handling
- b) Logging
- c) Session management,

The modules that were refactored for the Jasperreports component are those that implement the following crosscutting concern:

- a) Exceptional Handling
- b) Object retrieval
- c) Synchronization,



**Figure 3.18 - Architectural Diagram of Employee Management System**

For the proposed project, the following architecture was used.

- a) *The Client Side*: A client, such as amobile device, computer, laptop, or other device which, for presentation purposes, requests the resources via the internet using a user interface (generally a web browser).

- b) *The Admin Side*: The administrator is also a client—the mobile device, computer, laptop, and so on. which creates, updates, and deletes information by requesting resources over the internet using a user interface (generally a web browser).
- c) *The Web Server*: The server is where most web application operations take place. The safe interface, authorization and authentication channel with the browser are the responsibility of a specific application known as a web server. Whatever data the application needs is stored on a interactive database server.
- d) *The Application Server/Middleware*: The application server also known as middleware has the responsibility of providing the requested resources by contacting another server. PHP is a member of the middleware language family [123].

Such languages associate nearly with the Web server to interpret and process requests from the World Wide Web (www), collaborate with other server programs to complete requests, and then send the web server precisely what to serve to the browser of user.

The RUP development methodology was used to develop the Employee Management System; consequently, the system is implemented iteratively and incrementally. This will reduce the complexity of the system testing process by testing individual system components and evaluating expected outputs while developing the system. Thus, the testing technique is completed independently to the framework parts all along.

System testing can be broken down into two categories—blackbox testing and whitebox testing from the perspective of development.

- *Blackbox Evaluation*: a method of software testing that examines a software application's functionality without coding knowledge.
- *Whitebox Testing*: a method for testing software that examines an application's internal structure (codes).

Integration Testing is an additional method of testing.

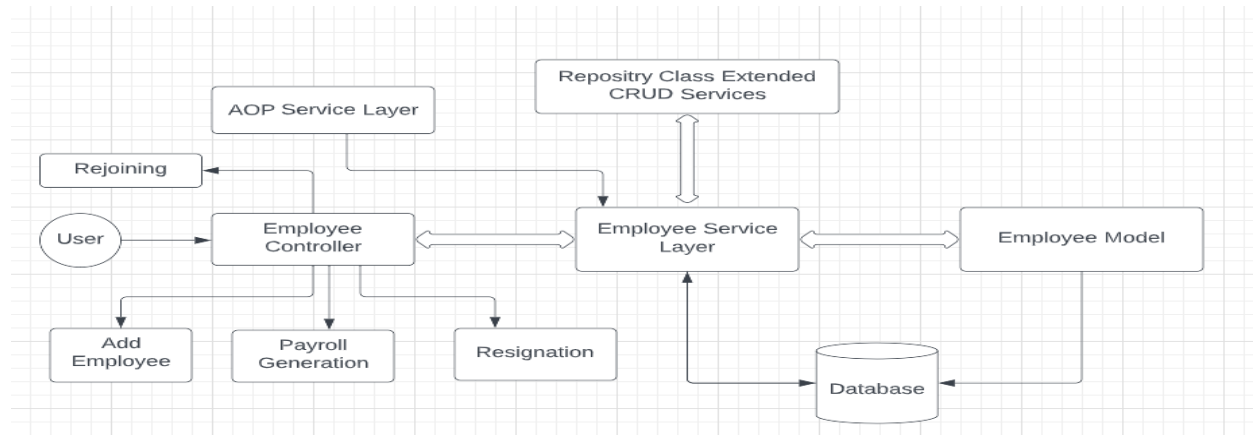
- a) In joining testing, individual programming modules are coordinated sensibly and tried collectively. Multiple software modules written by various programmers make up a typical software project. Data communication between these modules is the primary focus of integration testing. As a result, it is sometimes referred to as "Thread Testing," "I & T," and "String Testing" [20].

- b) **Regression Analysis:** A type of software testing known as regression testing is used to ensure that a recent program or code change does not adversely affect existing features. Regression Testing is nothing more than rerunning all or part of a set of test cases that have already been run to make sure that the functionalities are working properly. This testing is done to make sure that any new code changes won't affect the functions already in place. When the new code changes are complete, it ensures that the old code continues to function [21].
- c) **The Unit Test:** During application development (also known as coding), unit testing is performed. A section of code can be isolated and its correctness verified through unit testing. A unit in procedural programming may be a single procedure or function. The objective of unit testing is to demonstrate the correctness of each component of the software [22].

### **3.6 Research Work Flow Diagram**

Web Based Employee Management System is an entirely web-based automated system for managing employees in any business. The popular RAD (Rapid Action Development) Process Model was used to develop the entire software, which went through software requirement assessment, develop, execution, authentication, and maintenance in order [124]. It is possible to print reports for management and employees. It has the following notable characteristics:

- a) Obtain a variety of printable reports
- b) System for the managing leave for employees
- c) Daily attendance of employees
- d) Resignation of employees
- e) Transfer of employees
- f) Promotion of employees
- g) Adding employees
- h) System Manager for a certain department, title, workplace, position, bank, shift, etc.

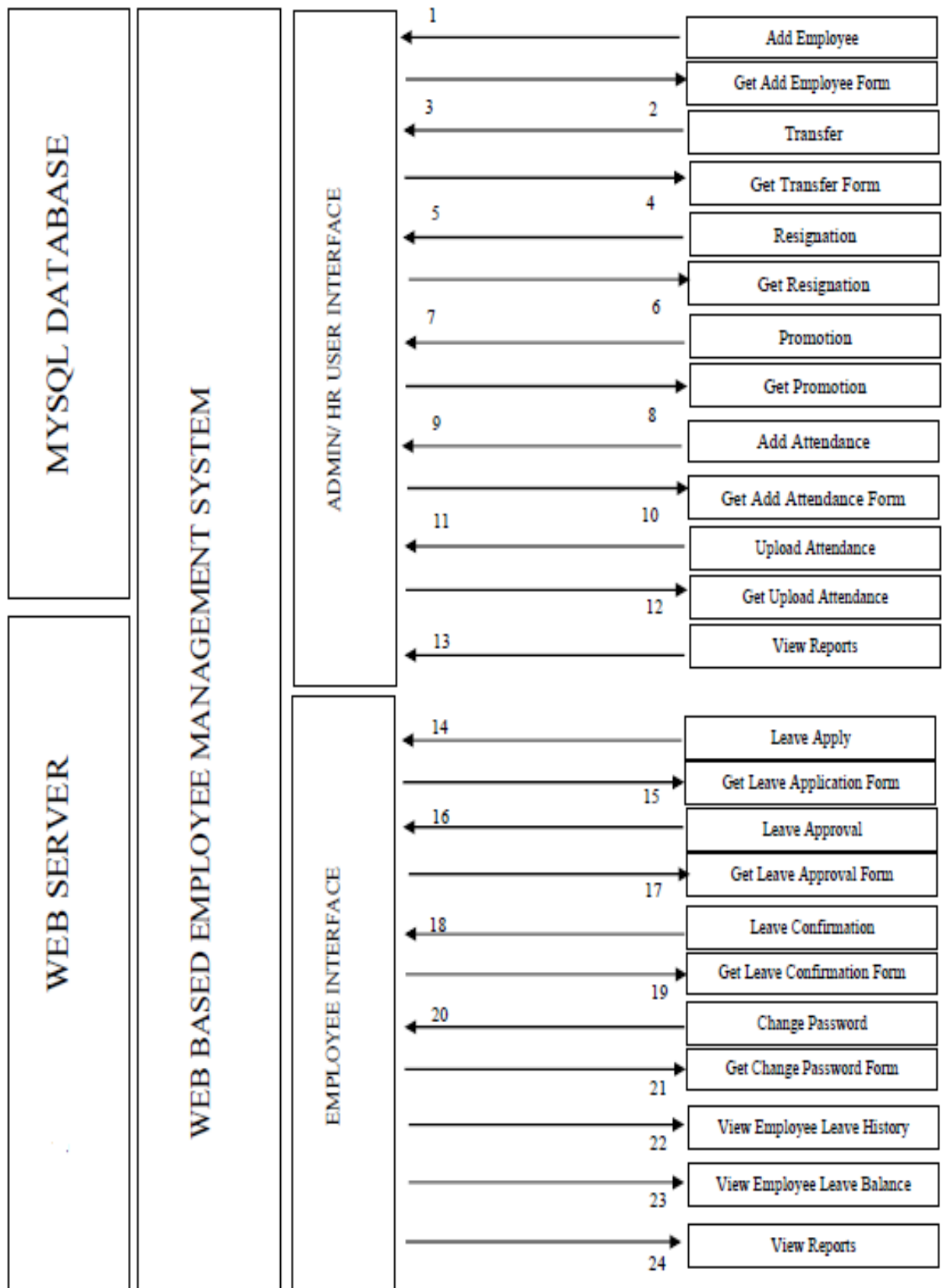


**Figure 3.19 – System Flow Diagram of Employee Management System**

In contrast to text-based interfaces, typed command labels, and text navigation, a graphical user interface (GUI) displays the user's options and information through graphical icons and visual indicators like secondary notation. Typically, the actions are carried out by directly manipulating the graphical elements [10]. The use of the visible language involves a number of fundamental principles sort out to furnish a client with a reasonable and reliable calculated structure.

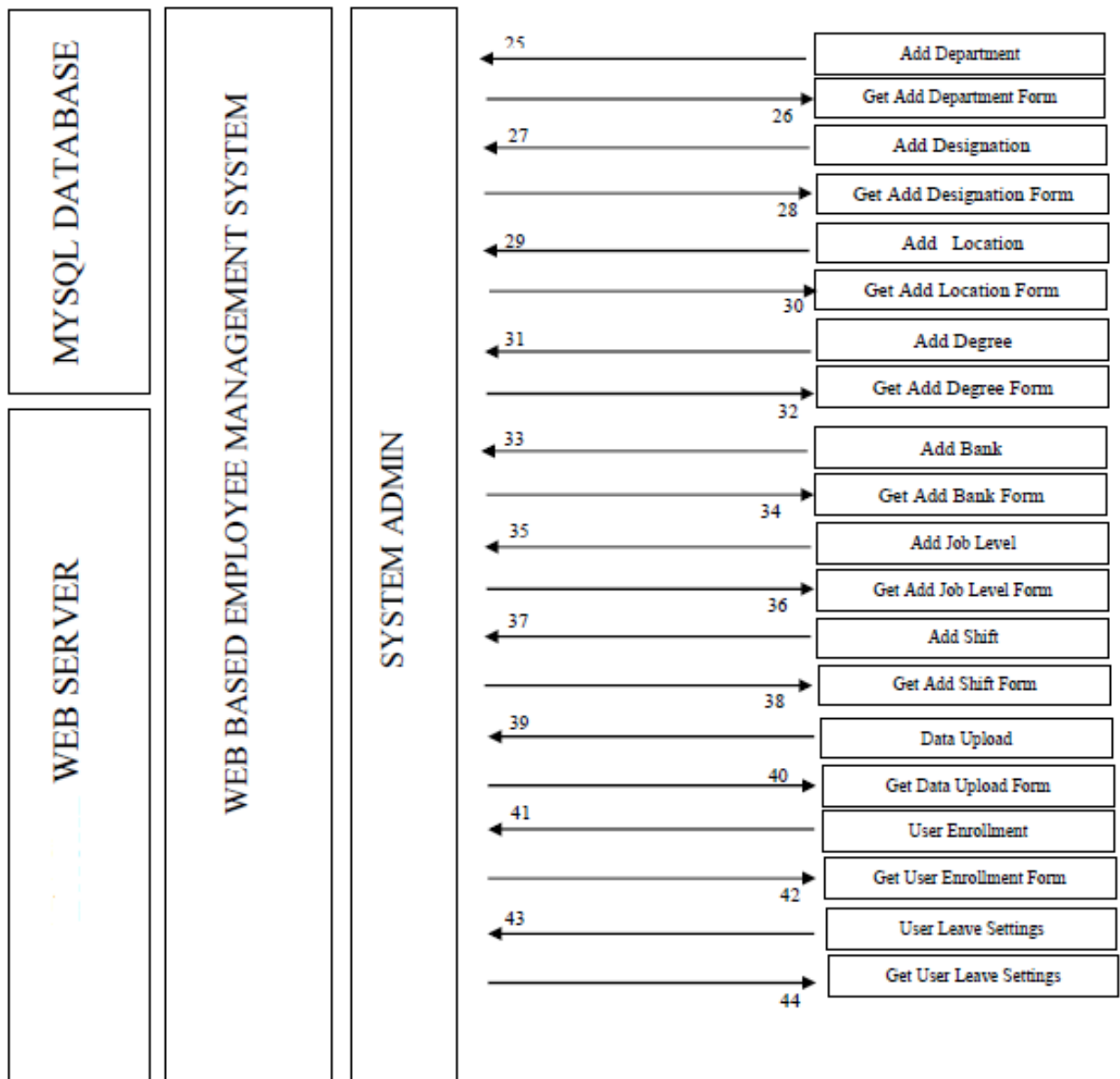
Regarding consistency, there are four perspectives, such as

- (a) *Internal for instance*: dialog boxes that are external, such as: text tool icons from the real world, like:-Signs from the real world:
- There should be no consistency;
  - Relationships: linking and disassociating items that are related can help achieve visual organization.
- (b) *Navigability*: there are main points of navigation;
- There should be an initial focus for the viewer's attention;
  - Direct attention to items that are important, secondary, or peripheral;
  - Assist in navigation throughout the material.
- (c) *Simplicity*: - There should be no confusion. It only includes the most crucial components for communication. Additionally, it ought to be as unobtrusive as possible.
- (d) *Express yourself*: -In order for a GUI to communicate effectively, it must maintain a balance between legibility, readability, typography, symbolism, multiple views, and color or texture. In this system, these guidelines have been used to create a GUI that is more user-friendly.



(a)





(b)

Figure 3.20 - Work Flow of an Employee Management System

### 3.7 Data Flow Diagram (DFD)

A DFD describes how data moves between system processes and data stores and external entities. DFD is a representation of the system's components and their modifications.

- DFD is a graphical method for expressing system requirements.

- The DFD lays out the system's needs as well as detects significant conversions that will become system design codes.

In system development, this is where the requirements specifications are broken down to the smallest detail. The procedure of defining a model's framework, programs, interfaces, as well as info to fulfill certain requirements is named as systems design. The application of system's concept to product development could be referred to as systems design. Systems engineering, systems architecture, and systems analysis share some similarities [125].

This software enables data manipulation by the Automation Application administrator. It makes it easier, safer, and more secure for the administrator to keep track of the data. The Employee Management System has two interfaces, which are outlined below:

**Administration Panel:** Since the software is connected to the database on the company server, no additional connections to other systems are required. During the course of this project's development, no system interface is required.

**Interface for Use:** The software must have a primary user interface and be designed as a web-based application. The main screen's format must be consistent and adaptable. The system needs to be easy to use. Pages must have consistent connections to each other. The system-assisted operations must be repeatable. PHP will be used as the dynamic scripting language for the web-based software. The software's front end will make use of HTML, jQuery, and CSS. In the back end, MySQL will serve as the database [126]. Additionally, printable reports will be generated using the fpdf library.

Database systems are made to handle a lot of information at once. Management of data includes defining information storage structures and proving information manipulation mechanisms. In addition, the database system must assure the security of the stored info in the face of attempted unauthorized access attempts and system crashes. If data are to be shared among multiple users, the system must prevent any potential anomalies from occurring.

A database management system, also known as a DBMS, is a set of programs that can access a collection of related data. The database, which contains information pertinent to an organization, is typically referred to as the collection of data. A database's primary objective is to provide a convenient and effective method for storing and retrieving database data [8].

The process to create an employee id is;

```
create table employee (emp_id integer not null auto_increment, emp_name varchar(255),
emp_department varchar(255), emp_salary float, emp_variable float, active varchar(200), primary
key (emp_id));
```

```
insert into employee(emp_name, emp_department, emp_salary, emp_variable) values ('Shrikant',
'TT', 100.00, 10.00)
```

Add Employee :

POST : <http://localhost:8080/add/employee>

```
{
    "empName":"Shrikant Patel",
    "empDepartment":"Accounts",
    "empBSalary":100.00,
    "empVariable":10.00,
    "active":"Yes"
}
```

=====

Get Employees :

GET : <http://localhost:8080/get/employee?id=1>

=====

Resignation :

PUT : <http://localhost:8080/resign/employee>

```
{
    "empId":1,
    "active":"No"
}
```

=====

Rejoining :

PUT : <http://localhost:8080/rejoining/employee>

```
{
    "empId":2,
    "active":"Yes"
}
```

=====

Payroll in AOP :

GET : <http://localhost:8080/payroll/employee?id=1>

=====

Generate Graph :

<http://localhost:8080/oop/generateGraphs?xAxisOOP=1000&xAxisAOP=500&xAxisValue=Program&yAxisValue=time&graphName=TimeTaken>

=====

Payroll in OOPs :

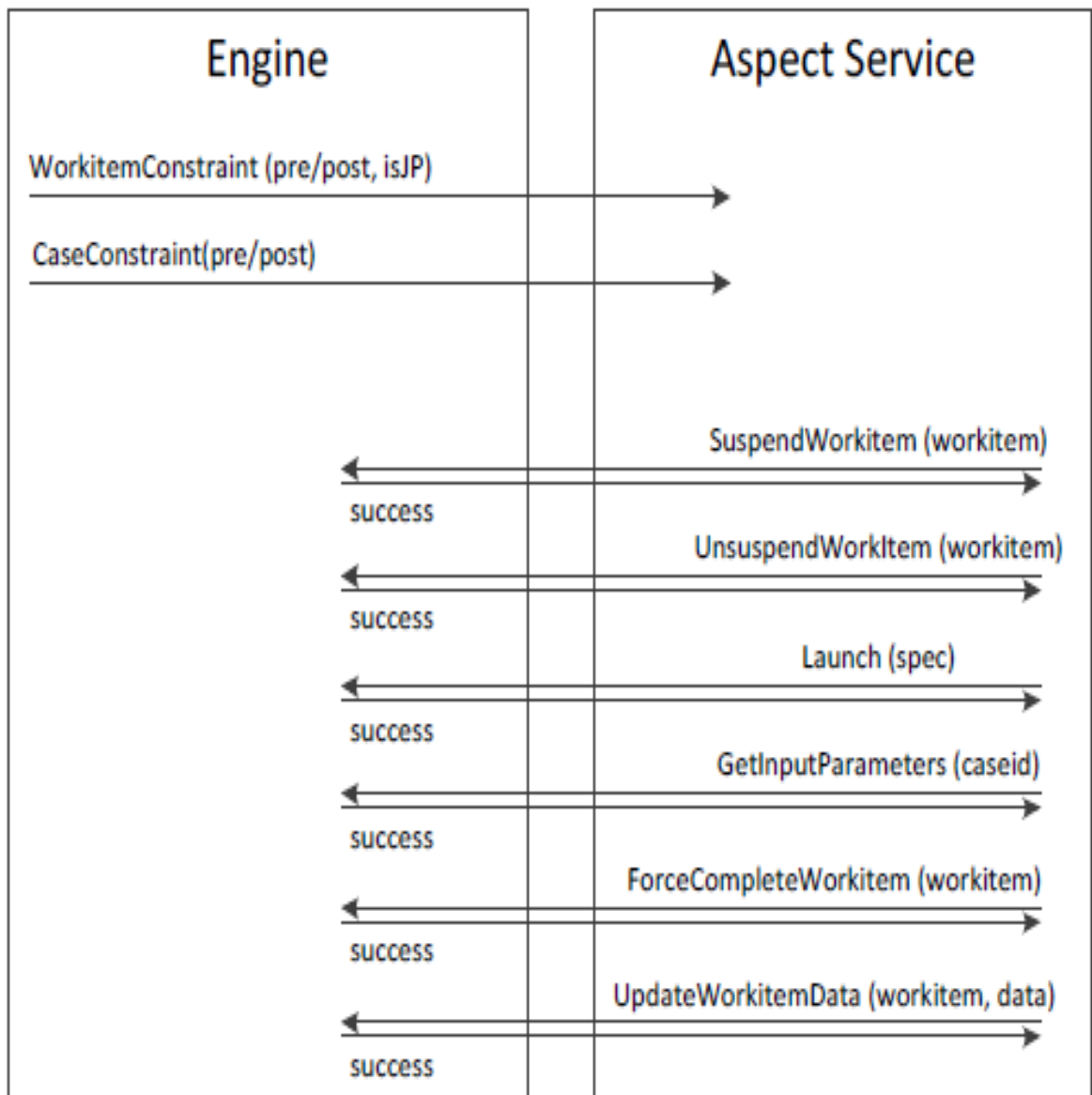
<http://localhost:8080/oop/payroll/employee?id=1>.

### 3.8 Aspect Interface

The various aspect service's required set of events and methods.

- In order to handle advised join points and the Proceed commands, work item components are utilized to capture required events. The definition of pointcuts for control-flow signatures is made possible by these constraints. The isJP parameter will be defined as a task extension that helps distinguish between the Proceed task and the advised join point task [127-130]. Another parameter tells whether the raised event is a postItemConstraint or a preItemConstraint of some kind [131].
- The aspect service is able to trace cases thanks to the CaseConstraint event [132]. It determines when each recommendation is finished. As a result, the core-concern could be resumed by aspect service to continue the process.
- The SuspendWorkitem is necessary because the aspect-oriented goal necessitates the suspension of the recommended join point and proceed tasks [133].
- The UnsuspendWorkitem is used to carry on a suspended workitem until the process is finished.
- A brand-new piece of advice is called "launch."
- A new advice's parameters can be obtained using GetInputParameters. Since the constraints should be read before the aspect is launched, this technique lets the service get the list of parameters.

- The ForceCompleteWorkitem method skips the Proceed task. The Proceed task ought to be carried out. It is, in point of fact, a placeholder for the advised join point. As a result, it ought to be forced to be finished this way.
- Both the advised task and the proceed task use UpdateWorkitemData to update the workitem data [134]. The data will be sent through the cross-cutting and core concerns. This matter was handled in this manner.



**Figure 3.21 Required Aspect Interface**

This chapter expelins the methodology of the research work, how the results we got after running the software and get the input and output for both the paradigm as in object-oriented as well as aspect-oriented.

# CHAPTER 4

## RESULTS & DISCUSSIONS

### 4.1 Mathematical Expressions Used

Employee Management System can be effectively modeled and implemented using Aspect Oriented Programming (AOP). AOP must be viewed in the context of current techniques like OOP in order to realize its full potential. In the abstract, it is frequently simple to appreciate a novel approach's dynamism; Practitioners still require as well as pursue policies to get initiated in the real world. When a project is under pressure, there usually isn't enough time to understand a technology and put it into perspective. Frequently recurring ambiguities regarding when to select an OOP solution over an AOP solution characterize initial AOP explorations. The enduring problems of assessing, developing, constructing as well as sustaining software systems are illuminated in novel ways by AOP.

Software components are typically not clearly separated in huge software systems. Instead, the lines of code that address a particular issue are typically dispersed throughout the programming and entangled with LOC that address a different issue. A concern's representation is dispersed throughout an artifact rather than being localized. If concerns are mixed together rather than separated, their representations are dispersed within an artifact.

Despite being distinct concepts, scattering and tangling frequently coexist.

According to the AOSD, "a structural correlation among representations of a concern" is the definition of crosscutting [135]. In this manner, it is like other types of structure, such as a block structure or a hierarchy. As a result, crosscutting is not the same as code tangling/scattering. While a crosscutting issue may be employed by code scattered during the code base and entangled with code that implements other issues, the presence of scattered/tangled program does not essentially indicate a crosscutting issue.

Instead, it could also be a bad way to program, as well as the issues at hand could have been simply referred by the modularization framework that was already in place.

AOP, like every other effective invention, pursues answers to issues that have been around for some time but have only recently come to light as software engineering struggles with greater complexity. Existing approaches have been and continue to be used to circumvent several issues for which AOP possibilities improved answers. When would the system of OOP-AOP offer better

returnswith respect to traditional OOP? is a common dilemma for designers and developers. Is it better to model an operation using a class or an aspect? A common topic that AOP novices are first introduced to is modularization of crosscutting concerns [136]. This is, in fact, a central idea of AOP, and it demonstrates how AOP is connected to some software engineering concepts that have been around for a long time. Crosscutting issues can be observed as behavior like instrumentation, security, exception handling, logging, and so on at a high level of abstraction. that extend across conventional responsibility distributions. Theseconduct is attained by specialized classes whose methods are appealed as needed in standard (i.e., non-AOP) OO executions.

There will be ten statements in a body of code where a Logger class's log method is called if logging is required at ten different locations. An approach to encapsulating dispersed functionality into modules is provided by AOP. Logging and other aren't the only supposedly minor issues that AOP addresses [137-138]. ManyOOconstructivesequences also have crosscutting configuration as well as can be employed using aspects in a reusable/modular manner. Aspects can be utilized to apply a Design by Contract style of coding. This is a central concern that all AOP clients face; as well as there are no prepared responses. Since programming evolved into software engineering, AOP concentrates on conditions that have been in the news. Since it became difficult to comprehend various aspects of the problem domain, SOCs as well as its standard have been of prime concern.

Vocabulary Size (VS), the number of class operations (OP), weighted operations in components/modules (NOA), the number of modules (NOM), non-commented lines of code (NCLOC), and lines of code (LOC) all play a significant role in determining the program's size in AOP.

**Table 4.1 - The Metrics Suite**

<b>Features</b>	<b>Metrics</b>	<b>Characterizations</b>
Coupling	Coupling Between Components	Counts the number of constituents that declare methods or fields that other components can call or access.
	Depth of Inheritance Tree	Counts how far down an aspect or class is declared in the inheritance hierarchy.
Size	Weighted Operations per Component	Counts the number of methods, suggestions, and parameters for each class or aspect.
	Number of Attributes	Each class or aspect's number of attributes is counted.
	Lines of Code	Counts the LOC.

	Vocabulary Size	Counts the system's classes, interfaces, and aspects as components.
Cohesion	Lack of Cohesion in Operations	Computes the number of method and advice pairs that do not authorize the similar instance variable to determine a class's or aspect's lack of cohesion.
Separation of Concerns	Concern Diffusions over LOC	Through the LOC, counts the number of transition points for each concern. The code's "concern switch" can be found at transition points.
	Concern Lines of Code	Counts the number of LOC whose primary function is to help implement a problem.
	Concern Diffusion over Operations	Counts the number of other methods as well as advices that authorize the methods as well as advices whose primary objective is to contribute to the execution of a concern.
	Concern Diffusion over Components	Counts the number of classes and aspects whose primary function is to assist in putting a problem into action, as well as the number of other classes as well as aspects that have authorize to those classes and aspects.

The mathematical expressions for used attributes are as follows:

- a) Size of the vocabulary:

$$\Omega = \Omega_1 + \Omega_2$$

Where  $\Omega$  = Program vocabulary.

$\Omega_1$  = number of unique operators.

$\Omega_2$  = number of unique operands.

- b) Length of the program

$$N = N_1 + N_2$$

Where  $N$  = Length of Program

$N_1$  = Total number of repetition of operators.

$N_2$  = Total number of repetition of operands.

- c) Volume

$$V = N * \log_2 \Omega$$

- d) Program Level

$$L = V^*/V$$

Where  $V$  = Volume of the program

$V^*$  = Potential Volume

$$0 \leq L \leq 1$$



Here,  $L=1$ , Since program is designed at the greatest possible level.

e) Difficulty

$$D = 1 / L$$

f) Effort

$$E = V/L = D*V$$

g) Estimated Program Length

$$\check{N} = \Pi_1 \log_2 \Pi_1 + \Pi_2 \log_2 \Pi_2$$

h) Required Time

$$T = E/S$$

Where  $T$  = Time needed for an effort of program

$S$  = Stroud Number (It is set to 18 for software experts)

$E$  = Effort

i) Estimated Program Level

$$\acute{L} = 2 \Pi_2 / (\Pi_1, N_2)$$

## 4.2 Results

A metrics suite was applied to the three MobileMedia implementations in Java, AspectJ, as well as EJFlow for the quantitative evaluation. Metrics for size, cohesion, separation of concerns, and coupling are included in this suite [11,22]. We chose these metrics because most of the empirical studies that have used them [5, 8, 18, 22, 25] have used them. Classic OO metrics serve as the foundation for the size, cohesion, as well as coupling metrics [11]. For facilitating the production of comparable outcomes, the original OO metrics were expanded to be put in a paradigm-independent manner. In addition, four brand-new metrics for enumerating SOCs are included in the metrics suite.

In our research, they compute the amount of which a solo model issue—exception handling—is planned to the design constituents (aspects/classes), operations (advice/methods), and LOC. A lesser value indicates a superior outcome for all of the attributes that are used. The attributes that are measured by each metric are linked to the metric's brief definition in Chapter 2. We have gathered the cohesion, coupling, and size metrics using our tool [17]. The concern metrics necessitated manual "shadowing" of the code, or determining that MobileMedia code section contributed to the problem with exception handling. The metrics are described in detail elsewhere [22].

The results of AOP and OOP are as follows;

TimeTakenAOP = 1546 ms

TimeTakenOOP = 11316 ms

Operators in AOP = 2

Operators in OOP = 5

Operands in AOP = 1

Operand in OOP = 3

Total number of repetition of operators AOP = 43

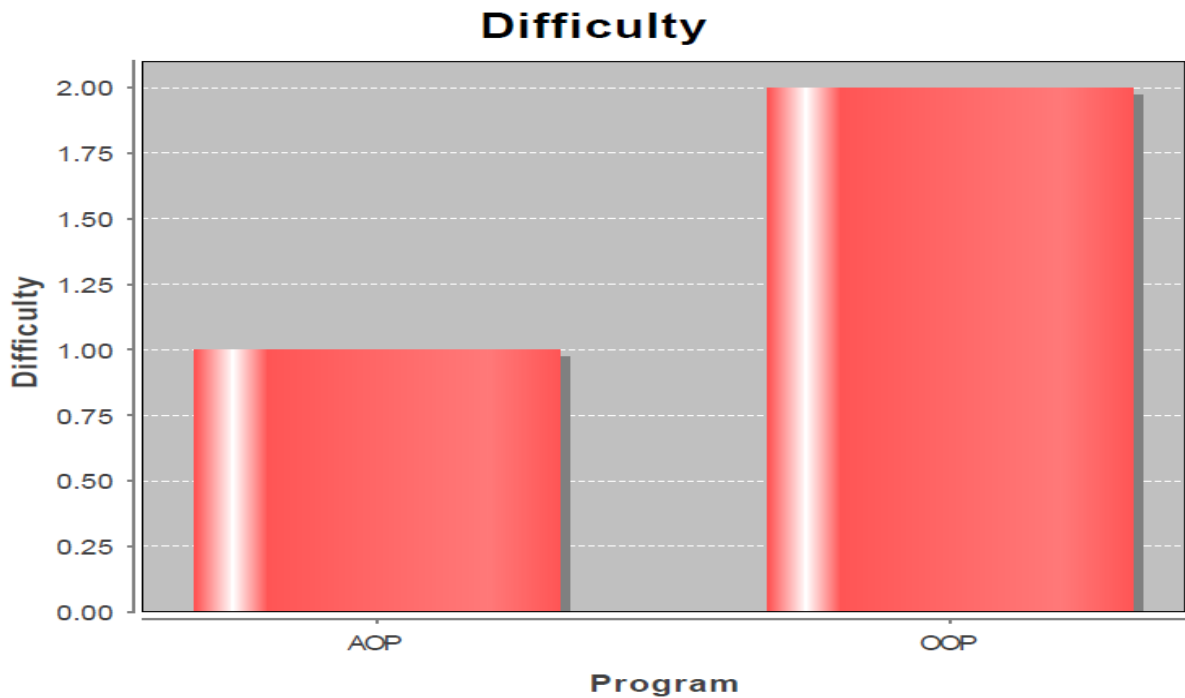
Total number of repetition of operators. OOP = 60

Total number of repetition of operands AOP = 58

Total number of repetition of operands OOP = 60

#### **4.2.1 Difficulty**

The two exception-associated issues of exception handling [8] and exception interface are the focus of this section's analysis of separation of concerns. The analysis of how the three case study solutions affect the concern measures is supported by the charts in this section. Figure 5.1 depicts the four concern measures used to separate the exception handling concern in Employee Management System. In the code of MobileMedia, the terms "try-catch," "try-finally," as well as "try-catch-finally" will be used to represent to definite aspects of the issue of exception handling. In the entire metrics for the exception handling concern, Java and AspectJ exhibit similar levels of scattering and tangling, as shown in Figure 4.1. Concern Diffusion over Components (CDC) is the metric for which the AspectJ version has the greatest value. This result was primarily caused by the fact that using AspectJ, it was impossible to completely modularize exception handling for some scenarios. The AOP system has low difficulty level as compared to OOP with the value of 1 in place of 2.



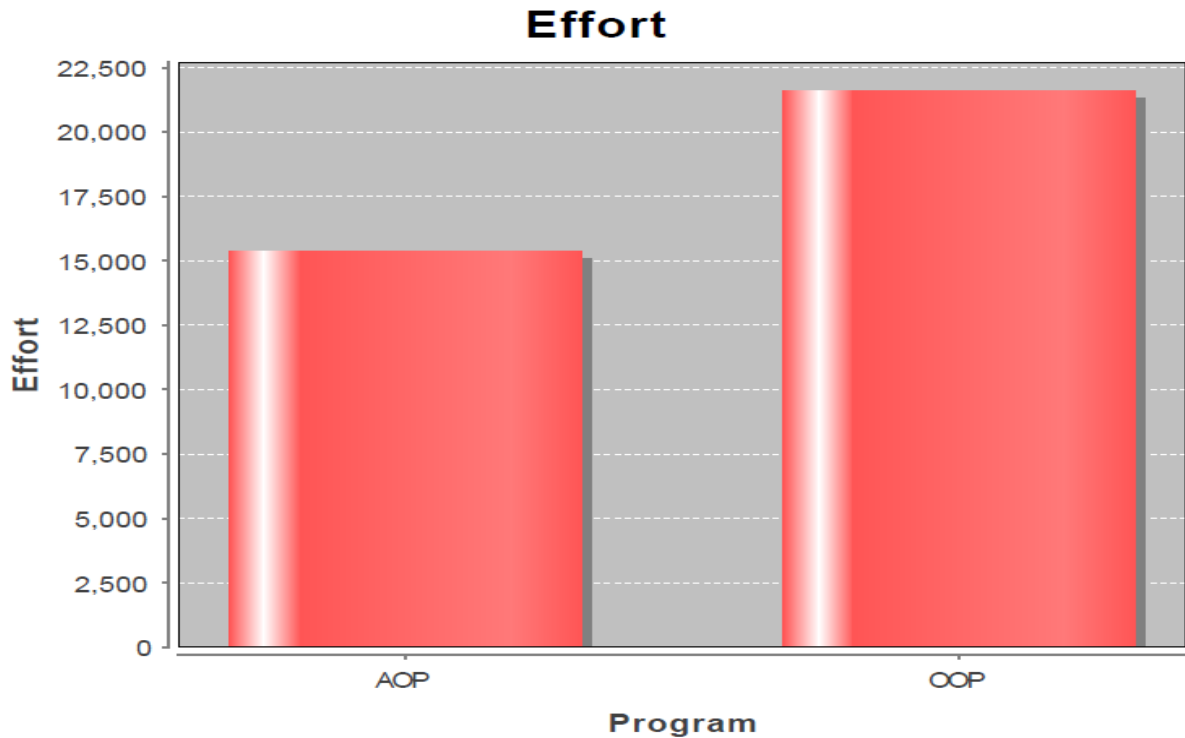
**Figure 4.1- Difficulty Level between AOP and OOP**

#### **4.2.2 Effort**

A controller using the Chain of Responsibility sequence [20] to handle the system's operations is shown in action in Figure 4.2. The connected controller calls its `handleCommand()` approach to observe if it can fulfill the request when the "Save Media" option is selected. The controller must return true in this case. The control is handed over to the subsequent controller in the chain if `handleCommand()` does not return false. In a similar vein, the handler that is a part of the method `getMediaInfo()` requires to return true for it to be able to skip any following calls that are related to successful saving operations.

Because the equivalent exception can be put by multiple requests of `getMediaInfo()` in numerous locations of the `handleCommand()` method, the entire try-catch block, consisting the request of `getMediaInfo()`, must be separated to an around advice in order to preserve the original handler's semantics. Because it keeps the abnormal/normal codes entangled in the extracted advice body, this solution is undesirable. In contrast, the ehandler advice code in EJFlow is outlined to address this issue. By extracting additional exception handling code from the application's constituents, this increases our method's applicability while maintaining the original handler's semantics.

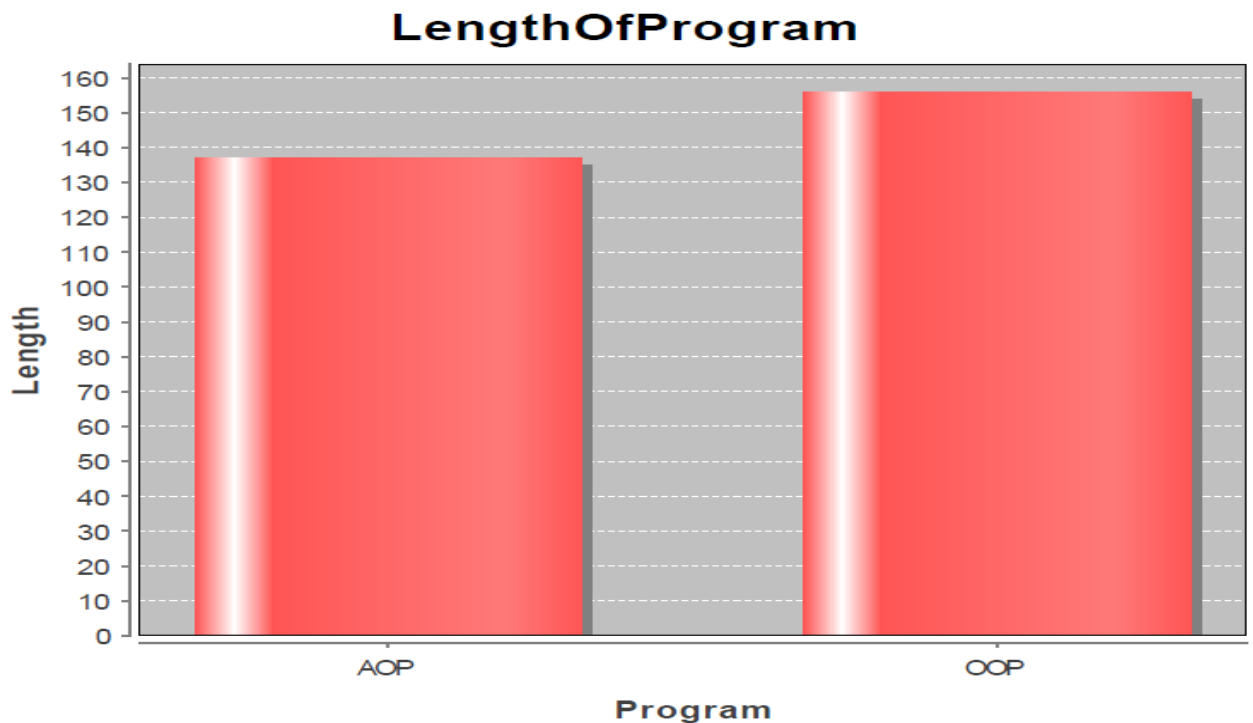
The handler that deals with exceptions that come through the channel InvalidMediaDataChannel, which was created by the Model component, is shown in Figure 4.2. The InvalidMediaDataException's propagation is captured by this explicit exception channel. The effort is low in AOP developed system as compared with the OOP model for Employee Management System. The effort level of AOP is 15,000, while for OOP is 21,750.



**Figure 4.2 -Comparison of Effort Level in between AOP and OOP**

### 4.2.3 Length of Program

Figure 4.3 displays the metric outcomes of the SOCs with respect to the length of program as exception interface concern in order to analyze the amount of attempt that can be prevented by utilizing EJFlow to handle the exception flow. With respect to AspectJ as well as Java executions, the measure outcomes showed a substantial lessening due to the use of explicit exception channels. With respect to CDC, explicit exception channels reduce the number of components in the exception interface code. As a result, adaptations in a channel can be simply made by only looking at six constituents, as opposed to the Java and AspectJ implementations' respective totals of 10 and 14. Additionally, the amount of code required to define the exception interface (CDO and CLOC) is reduced by EJFlow..

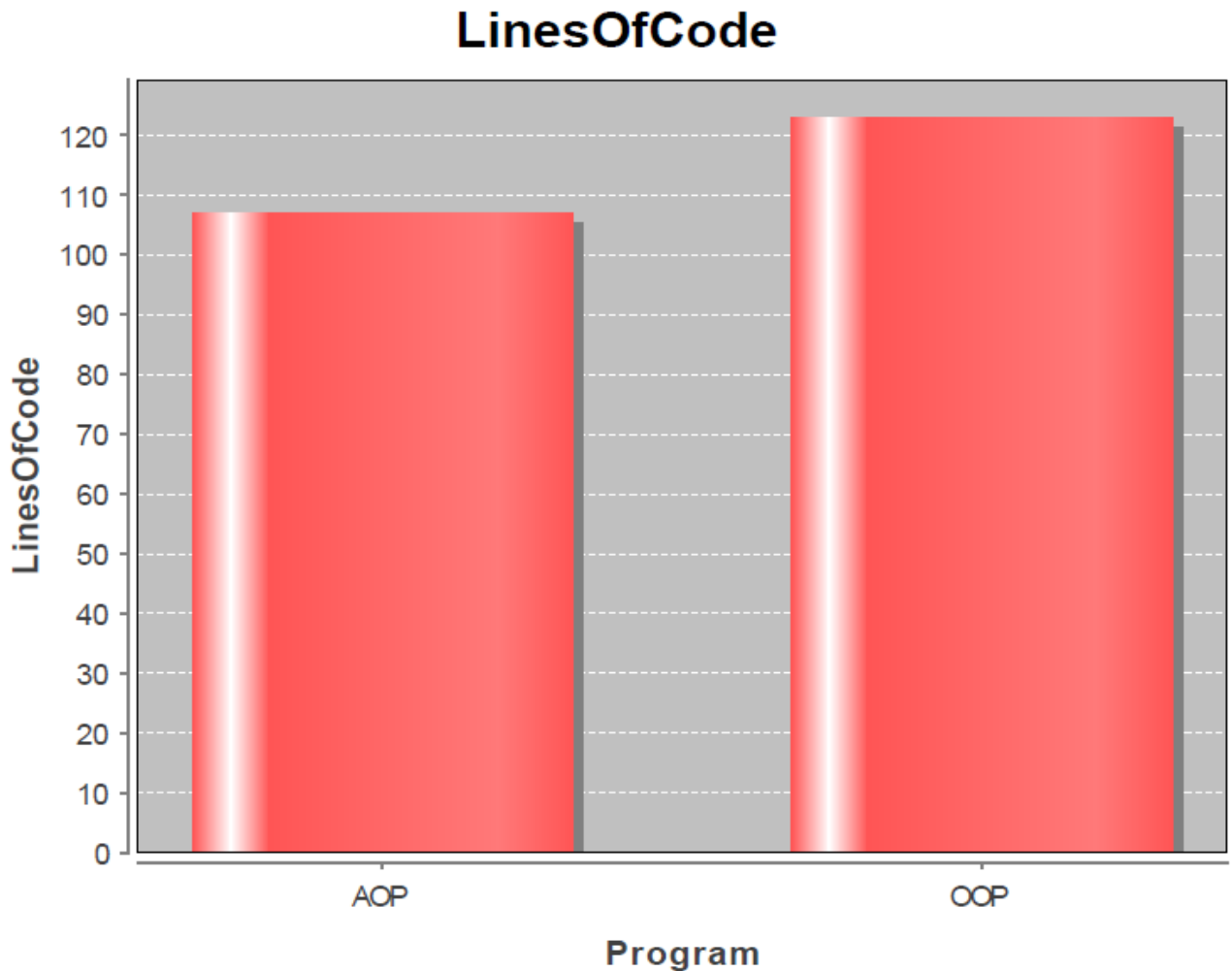


**Figure 4.3 - Comparison of Length of Program between AOP and OOP**

### 4.2.3 Lines of Codes

CLOC as well as CDO obtained results were superior for the MobileMedia refactored EJFlow version. The way EJFlow modularized handlers that were only responsible for remapping caught exceptions was directly responsible for the reduction in the operations number as well as LOC that contained exception handling code. Because they enforce the exception interface among constituents, these handlers play a crucial role in the Java implementation. To modularize these handlers in AspectJ, the code is moved to after throwing advice.

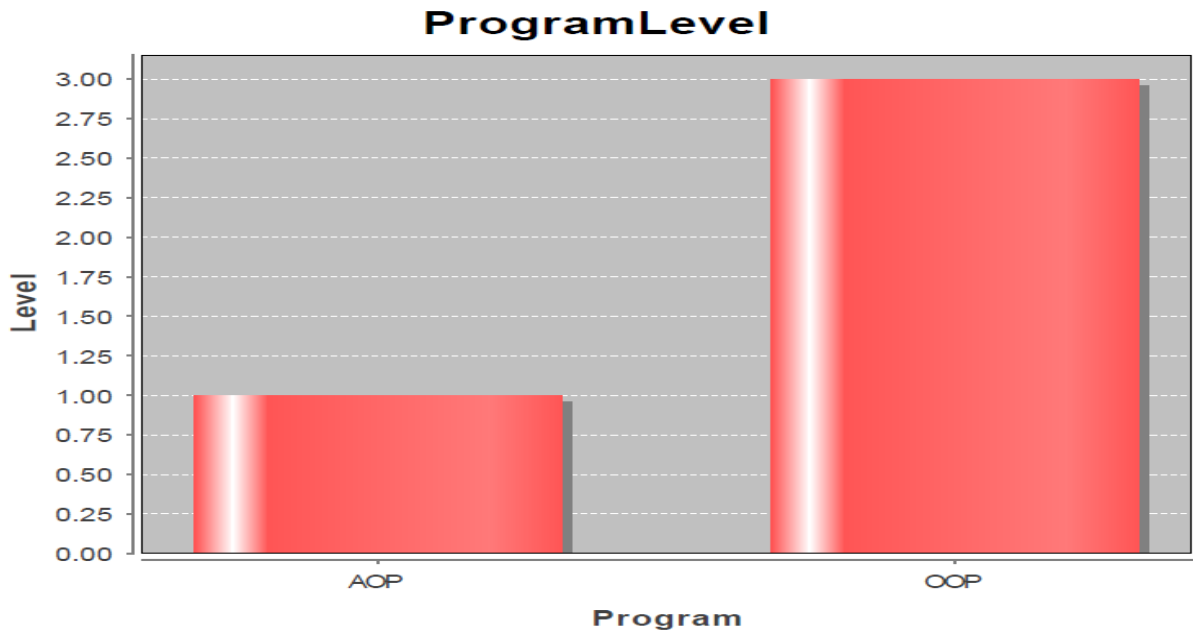
However, in contrast to Java's try-catch blocks, the AspectJ implementation's CDO and CLOC values increase with each handler advice. Additionally, the greatvalue of CLOC in the AspectJ solution is a result of the overhead associated with extreme utilization of AspectJ constructs like declare soft and pointcut. Additionally, we believe that this makes the program more difficult to comprehend. To enforce the exception interface, EJFlow makes use of the declare interface. As a result, two lines of interface declaration can take the place of a significant amount of code that is just used to execute these users. Once remapping handlers account for nearly 30% of all users in actual Java applications, this effectiveaspectization has aoptimistic impact on any system's overall result.



**Figure 4.4 -Comparison of Line of code between and AOP and OOP**

#### **4.2.4 Program Level**

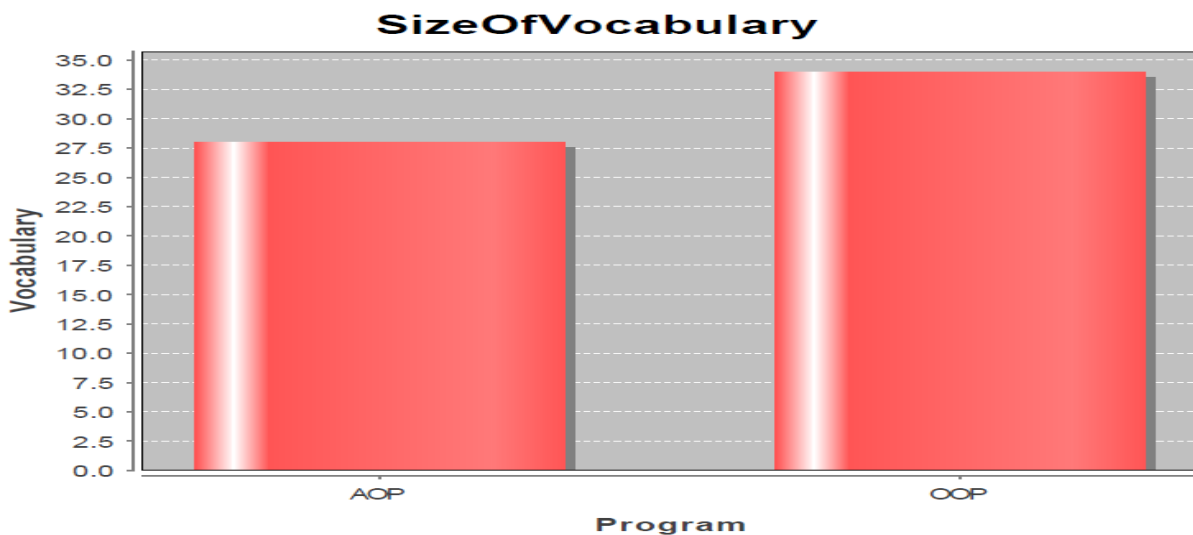
The empirical data shown in Figure 4.5. Our assertion that code reuse is facilitated by a universal perspective of exception flow. Most methods handle exceptions within the similar interface in the same way. As a result, the developer will have an easier time determining which handlers can be reused. In the EJFlow version of MobileMedia, the coupling among normal behavior as well as error handling (CBC) is also lessen. This usually exists due to the approaches in an interface don't have to list the exceptions they signal in their interfaces explicitly. The number of classes on which such approaches rely will unavoidably decrease because several of such approaches do nothing with these exceptions other than indirectly propagate them. The level of programming is high in case of OOP as compared to AOP due to which the difficulty level of OOP becomes high.



**Figure 4.5 - Comparison of Program Level between and AOP and OOP**

#### 4.2.5 Size of Vocabulary

By removing exceptions that are only used to enforce the exception interface between components, EJFlow also makes it possible to reduce the number of components. For instance, the exception interface is not enforced by any exception in Figure4.6. As a result, the exceptions `UnavailableMediaAlbumException` and `AlgorithmicException` can be removed from the application if they are not used in any other context. As a result, there are fewer lines of code and exception classes (as measured by the Vocabulary Size metric).

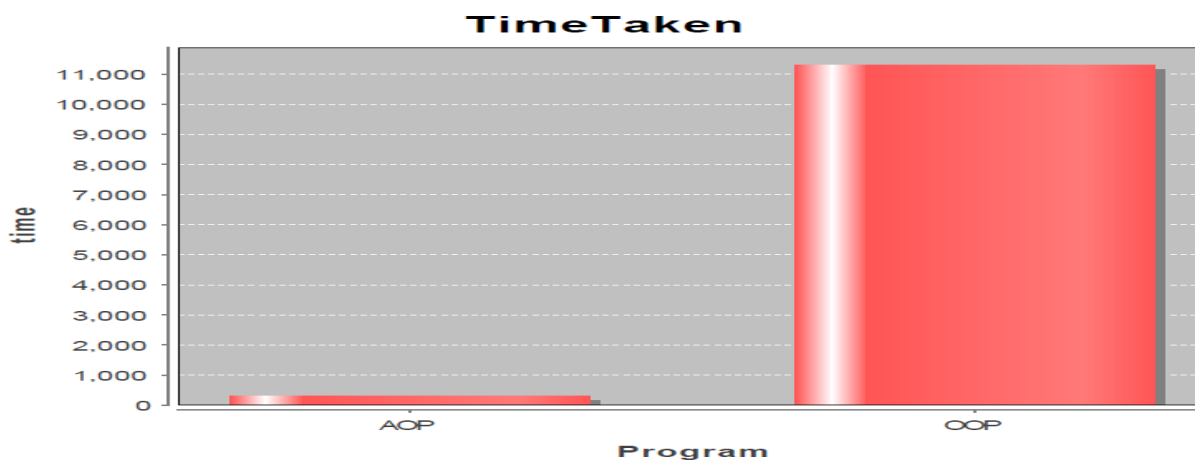


**Figure 4.6 - Comparison of Size of Vocabulary between and AOP and OOP**

## 4.2.6 Time Taken

Lippert and Lopes utilized a deep-rooted AspectJ version to refactor exception handling code in a big OO architecture known as JWAM to aspects in a seminal study [28]. The purpose of this research was to determine whether aspects for splitting exception handling code from other application code were useful. The utilization of aspects to modularize exception handling/detection, according to the authors, has numerous advantages when applied to a reusable structure that executes general (i.e., nonapplication explicit) error handling strategies. These advantages include improved reuse, low interference in program texts, as well as a reduction in LOC number. The objective of Castor Filho et al.'s corresponding empirical research work [8] was to comprehend the advantages and disadvantages of utilizing aspects to structure error handling code in practical applications.

Application-specific exception handling policies were implemented by four distinct systems, three of which were OO as well as one AO, in this investigation. It involved refactoring the four applications' error handling code into aspects. According to the study, some of the conventional wisdom regarding the utilization of aspects to modularize exception handling only applies to straightforward situations that are not always feasible. For instance, the researchers discovered that reusing non-trivial exception handling program is a challenging endeavor that is contingent on a number of factors. The use of aspects to assist in tracking the flow of exceptions within a code is not suggested by any of the aforementioned studies. In addition, despite the fact that both of them noted some of AspectJ's limitations when it comes to handling exceptions, they did not suggest appropriate language extensions to address such issues. As shown in Figure 4.7, the time taken by the AOP model takes much lesser time in comparison to the OOP model.



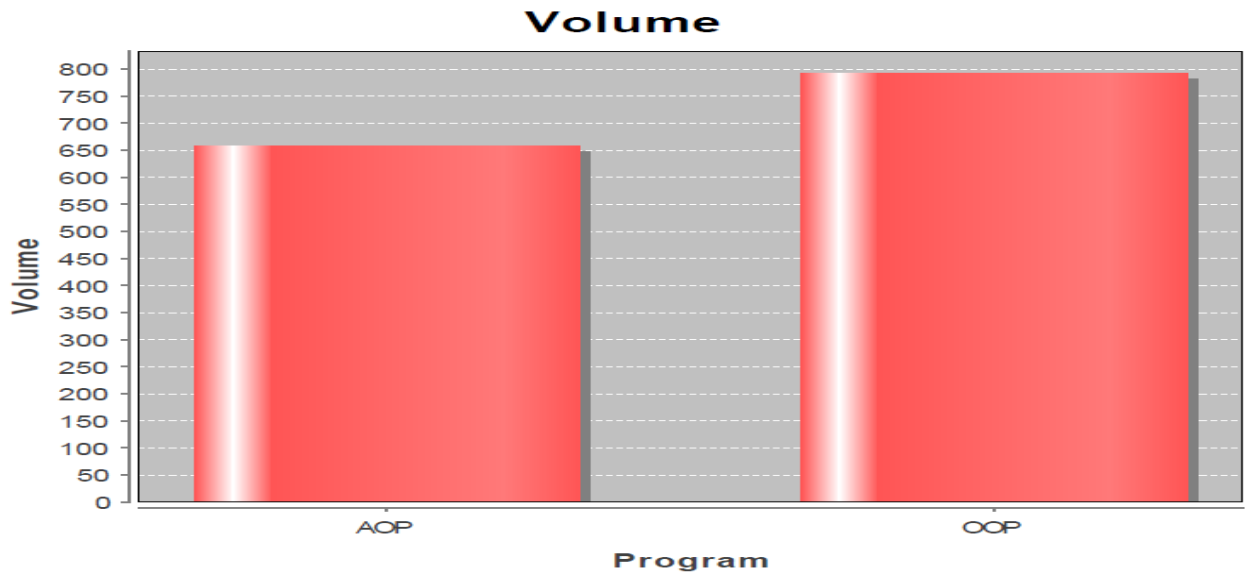
**Figure 4.7-Comparison of Time Taken between and AOP and OOP**



### 4.2.7 Volume

As the organization grows, the data also grows which directly affects the volume of program. It is important to run efficiently even if the programming has high volume. In the same database, the volume occupied by AOP model for Employee Management System is much lesser than the OOP model. Figure 4.8 clearly provides the detailing of the developed systems. This work performed maintenance tasks in order to evaluate the unpredictability of both the AOP/OO forms of the two constituents. As previously stated, the outline of a novel aspect to assess the implementation time of SQL statements is represented to as a maintenance task. The substitution of the logging constituent with a different constituent with the similar operation is referred to as another maintenance task. A feature that allows for profiled implementation of SQL statements with respect to the database was added during this maintenance task; Specifically, observe the execution time of these statements. The following code-level changes were counted to determine the influence of variation or changeability:

- a) Number of lines of code modified (LM)
- b) Number of lines of code removed (LR),
- c) Number of lines of code added (LA),
- d) Number of operations modified (OM),
- e) Number of operations removed (OR),
- f) Number of operations added (OA),
- g) Number of modules added (MA),
- h) Number of modules modified (MM),
- i) Number of modules removed (MR).



**Figure 4.8 - Comparison of Volume between and AOP and OOP**

### 4.3 Assessment of Metrics

The six compositions were measured using a metrics suite for the quantitative evaluation. SOCs, cohesion, coupling, and size are all important modularity dimensions that can be captured by these metrics in pattern compositions. In order to be applied paradigm-independently, the coupling, cohesion, and size metrics support the production of comparable outcomes among AsectJ as well as Java solutions. They are extensions of conventional as well as OO metrics. New metrics for evaluating concern separation are also included in the metrics suite. The amount to which a single system concern corresponds to the design constituents (aspects/classes), operations (advices/methods), as well as LOC is measured by the separation of concerns metrics.

Our own measurement tool was used to collect some of the data during the measurement process. Except for the metrics of concern separation (CDO, CDLOC, and CDC), it supports all metrics. The shadowing of each aspect, interface and class, in both executions of the pattern compositions preceded the data collection for the SOCs metrics. The pattern roles that they use were used to shadow their code. In order to examine its crosscutting framework in pattern compositions, we treated every design pattern as a concern. The information for the SOCs metrics (CDC, CDO, and CDLOC) were manually gathered following the shadowing. This work concentrates on the more pertinent outcomes because of the limited space. The entire description of the collected data can be found elsewhere.

We have noticed that the attributes of cohesion (LCOO), complexity of operations (WOC), and number of attributes (NOA) were also associated with the structural group as well as the included patterns, as shown in the preceding sections. Because AspectJ lessens the overuse of inheritance concepts, the AO solutions were generally better as NOC measures. However, as shown in Figures 4.2, 4.3, and 4.4, the majority of measurements indicated that AspectJ implementations produced more LOC as well as higher coupling (CBC) than Java implementations. Although, an in-depth examination of the implementations reveals that the presence of generic aspects in a number of AspectJ pattern executions was linked to the higher CBC as well as LOC values for AO solutions in a number of instances. These aspects aim to increase the pattern solutions' reusability. The artificial occurrence of generic aspects has artificially resulted in greater values for CBC/LOC in several of the compositions that were investigated. These compositions had some contestant classes playing the pattern roles.

When we compared the composition examples found from the agent-based application as well as the measurement tool with those obtained from the middleware implementation, this effect was more pronounced. The first application features typically only had a small number of participant classes, whereas the second ones typically had a large number of participants. For instance, the measurement tool's composition Decorator with Bridge has promising CBC/LOC values for the AspectJ execution. However, it is essential to point out that, in a number of instances, a greater CBC value was in fact a clear indication of greater AspectJ solution coupling. In some invocation-based compositions, for instance, coupling issues arose when inter-pattern invocations were unavoidably transported to the aspects' code. The inter-aspect dependencies /aspect-class introduced novel coupling sources in the composition execution due to the implicit association among the base classes. Compositions with intra-method interlacing as well as overlapping revealed additional issues related to coupling.

#### **4.4 Threats to Validity**

The research work provides the outcome that is promising because it shows that the AO method can help improve maintainability. This is consistent with other research work of a similar nature that have been carried out in the past. However, before we attempt to generalize the findings of our work, we must exercise caution.

- a) Because the sample size for our case study is relatively small, it would be challenging to envisage the same outcome for a very large system. Furthermore, in our work, the AOP language—AspectJ—was used to refactor only a few crosscutting issues from the OO

versions of Jasperreports as well as OpenBravo. It would be helpful to have an entire AOP version of the OpenBravoPOSmodel in order to compare how easy it is to maintain the two systems. In addition, the fact that our evaluation of maintainability is depending on a single system makes it challenging to apply results to other systems. For a more reliable generalization, additional experiments on several models from many areas are required.

- b) The fact that the individuals who measured the system's maintainability were also responsible for the refactoring of components' aspects is another issue related to our case study. There may be a bias as a result of this. However, we made every effort to limit the refactoring so that the goal was not to make the system easier to maintain but rather to refactor the aspects.
- c) The evaluation of changeability lacks sufficient depth. Even though our methods were comparable to those of other works, the kinds of variations still require development. Invetsigators et al., for instance provided modifications that could be made at the component and system levels at the code level. Furthermore, it specifies the least number of random variations that must occur. This method will be utilized in our subsequent work.
- d) Despite the fact that calling a local module requires less effort than calling an external component, the organizational complexity metrics chosen measured each LOC to be identical.

For AOP-based systems, this is a fascinating phenomenon because the maintenance designer will need to learn how to call an AOP constituent. In future studies, this outcome of the calls to an AOP constituent will also be examined.

This chapter explains the results of the research work with the output of the different metrics of object-oriented and aspect-oriented paradigm.

## CHAPTER 5

### CONCLUSIONS & FUTURE DIRECTIONS

#### 5.1 DISCUSSION

At the moment, we have established a connection among code reusability as well as the most common software metrics. We demonstrated this on three object-oriented-designed, complex software projects as well as discovered that our model considerably calculates code reusability for any level of complexity, even when it is extremely indefinite. We demonstrated, employing a mathematical and stochastic Markov Modeling approach, that our model can extract more info about code reusability as uncertainties rise. In software engineering, design patterns are very important. Since more than a decade, the IT industry as well as software project creators have been seeking consultation to reduce production costs in response to rising customer demands. Code reusability becomes the most noticeable method for reducing costs and necessitates the decision of a highly skilled technical architecture.

Choosing which parts of the code should be kept the same and which parts should be designed from scratch are the two difficult aspects of code reusability.

In the first difficult part, a creator can simply resolve which part of the program needs to be kept depending on the user's needs. Although, the problematic portion is deciding what to do about the new program that needs to be constructed from the ground up. The novel codes set that requires to be coded is typically created to have a definite code level reusability for the unpredictable future user, depending on an experienced architecture. In this instance, an impractical design will cause production to cease completely as well as may not fulfill the reusability requirement for new projects. As a result, the estimation of code reusability in complicated software projects will be the primary focus of our future research. We anticipate that the stakeholder will strongly endorse our design concept as the most cost-effective tool to date.

In order to produce the AO versions of the first two systems, we used AspectJ to reengineer the existing Java implementations. We have reutilized both prevailing Java as well as AspectJ executions for the third case study. We have attempted to optimize the splitting of every sequence from the second arrangement in the grouping as well as application-specific concerns in both the OO and AO solutions. With this strategy, our goal was to make the configuration employment as modular as possible to make it easier to mix and match different patterns. In order to link this study's findings with those of previous ones, we have also tried to keep as much of the original use

of the configuration employments as possible while implementing the AspectJ versions. However, in order to attain the proposed pattern modularization, we had to make some minor adjustments to the original AspectJ executions due to the unique characteristics of each application. In addition, we had to rely on a different AspectJ version in other instances as the application context required a particular pattern variant. The aspectization of some patterns was caused by certain compositional circumstances, as will be discussed throughout the paper. It ought to be looked at as a bad result for the AspectJ solution.

We had to make sure that both Java and AspectJ versions were executing the identical operations for comparing the two executions of the structures. As a result, a few minor adjustments were made to the patterns' code. Changes of this nature included:

- a) To ensure that the composed patterns' aspect-oriented (or object-oriented) implementations are equivalent between the two versions by adding or removing a functionality, such as a method, a class, or an aspect; We evaluated the functionality's relevance to the pattern implementation before deciding whether to integrate or disintegrate remove it from the implementation; and
- b) Due to the granularity of our metrics, we wanted to make sure that both versions utilized the same coding styles.

## 5.2 CONCLUSION

For two decades, software engineers have been focused on determining how good software is. The AO paradigm emerged as a consequence of the separation of concerns issue. Due to its close ties to object-oriented programming, this new paradigm raises quality concerns. We will be able to construct quality models and evaluate the quality of aspect-oriented programs thanks to our work on the impact of AOP on object-oriented metrics and the implementation of a measurement framework. Empirical research will be used to verify the proposed work. In point of fact, we will be capable to evaluate the quality of an OO program prior to and following AOP program changes using our framework. While keeping track of exception control flows, our novel exception handling mechanism makes use of AOP methods to endorse better separation among error handling code as well as normal programming. We also want to reduce the number of common issues that arise when traditional exception handling mechanisms are used and have an effect on software system quality as a whole. We assert that the proposed model has three primary advantages:

- a. Without having to look at other parts of the program, it makes exception flow explicit as well as understandable locally;

- b. It improves error handling code reuse, enhancing program modularization;
- c. By splitting the handlers and obnoxious exception interface declarations, it makes normal and error handling code easier to maintain.

We have employed the majority of EJFlow, with minor syntactic additions to AspectJ. Our current research includes an empirical comparison of EJFlow's error proneness to the conventional suggestions deliberated in this work. Additionally, we aim to overcome a challenge of our assessment by evaluating EJFlow's scalability in software evolution as well as maintenance situations.

### **5.3 FUTURE DIRECTIONS**

The systematic review has shed light on the most recent developments in coupling measurement for AOP. As a result, the requirement for specific AOP construct-specific fine-grained metrics has been brought to light. As a result, key contributors to maintainability may be overlooked by commonly used existing metrics. 200 As a result, current AOP maintainability studies make use of coupling metrics that-

- a. Can be used with confidence with a wide range of AOP languages,
- b. Differentiates among the several coupling dimensions, and
- c. Takes into account specific language constructs.

Additionally, we've noticed that static coupling metrics get a lot of attention in AOP maintainability studies. In some of the examined studies, dynamic coupling metrics [1] for AOP have not been utilized. Given that many AO composition mechanisms rely on the behavioural program semantics, this came as a surprise. Additionally, important maintainability characteristics like error proneness are never explicitly evaluated. It's not easy to validate new metrics. Kitchenham mentioned the difficulty of validating metrics solely using predictive models [29]. Metrics might not be appropriate indirect measures of maintainability without theoretical validation. As a result, even AO metrics derived from OO metrics that have been empirically validated may not be theoretically sound predictors of maintainability. In fact, some AO metrics violate the representation condition and other criteria, as our systematic review revealed.

New mechanisms for modularly implementing cross-cutting issues are proposed by aspect-oriented programming. Aspect-oriented programming is evaluated in this paper in relation to four interconnected issues that arise across three FreeBSD kernel versions. Aspect-oriented programming has the potential to improve the evolvability of OS code, but there are still many

unresolved issues, as evidenced by the costs of our AspectC prototype, the support aspects provided for evolution, and the ways in which aspects enabled us to make these implementations modular. In this section, we briefly discuss the study's limitations, the aspects' generalizability, and possible future experiments.

Programming is seen by software users as a tool that can be used to help them operate together in their specific Structure. A number of qualities make up the structure of quality. As a result, a system that demonstrates the features as well as their links typically captures quality. The examples are useful; They demonstrate what people consider to be significant when discussing quality. Based on the AOSD worldview and product item quality detail, extraordinary associations employ distinct quality models. A mapping is constructed using REASQ among the ISO/IEC guidelines and the emerging AOSD discipline. One of the main goals of AOSD is to find connections between non-practical concerns and quality requirements and at least one quality attribute of the standard quality model (potential cross-cutting concerns). The manner in which a perspective addresses the issue of crosscutting concerns (given that these are well-renowned) by typifying them in a specific configuration via an arrangement constituent is generally agreed upon. Through a structure table, the method is accessible from the beginning of the product development process, demonstrating framework engineering to encourage the plan as well as implementation phases, for example.

According to the researchers, the structure of programming is controlled by its internal characteristics. This makes it easier for product engineers to achieve the programming's external features, which include convenience, reusability, reliability, portability, practicability, integrity, flexibility and accuracy.



## REFERENCES

- [1] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. (2001). An Overview of AspectJ. *Proceedings of the 15th European Conference on Object-Oriented Programming*, Springer-Verlag, 327–355.
- [2] Laddad, R. (2010). *AspectJ In Action*. 2nd Edition, Greenwich: Manning Publications Co.
- [3] Colyer, A., & Clement, A. (2005). Aspect-oriented Programming with AspectJ. *IBM System Journal*, 44(2): 301-308.
- [4] Weiser, M. (1982). Programmers Use Slices With Debugging. *Communications of the ACM*, 25(7): 446- 452.
- [5] Ishio, T., Kusumoto, S., & Inoue, K. (2004). Debugging support for aspect-oriented program based on program slicing and call graph. *Proceedings of 20th IEEE International Conference on Software Maintenance*, 178–187.
- [6] Xu, G., & Rountev, A. (2007). Data-flow and Control-flow Analysis of AspectJ Software for Program Slicing. *Technical Report OSU-CISRC-5/07-TR46*, Computer Science and Engineering Research Center, Department of Computer Science and Engineering, Ohio State University.
- [7] Zhao, J. (2002). Slicing aspect-oriented software. *Proceedings of 10th International Workshop on Program Comprehension*, 251–260.
- [8] Mohanty, S. R., Behera, P. K., & Mohapatra, D. P., (2015). Slicing Aspect-oriented program Hierarchically. (*IJCSIT*) *International Journal of Computer Science and Information Technologies*, 6(6), 5004–5013.
- [9] P. Naur and B. Randell, editors. *Software Engineering: Report of the Working Conference on Software Engineering, Garmisch, Germany, October 1968*. NATO Science Committee, 1969.
- [10] E. W. Dijkstra. The Humble Programmer. *Communications of the ACM (CACM)*, 15(10):859–866, 1972.
- [11] E. Figueiredo, C. Lucena, and A. Garcia. AJATO: An AspectJ Assessment Tool. *In Demo Session of the European Conference on Object- Oriented Programming (ECOOP)*, 2006.
- [12] B. Boehm. *A View of 20th and 21st Century Software Engineering*. *In Proceedings of the International Conference on Software Engineering (ICSE)*, pages 12–29. ACM Press, 2006.

- [13] M. Jackson. The Structure of Software Development Thought. *In Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 228–253. Springer, 2006.
- [14] The Standish Group. Chaos Report. Technical report, Standish Group International, 2003.
- [15] R. L. Glass. The Standish Report: Does it Really Describe a Software Crisis? *Communications of the ACM (CACM)*, 49(8):15–16, 2006.
- [16] E. Yourdon and L. L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press, 1979. copyright 1979 by Prentice-Hall.
- [17] D. L. Parnas. Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Engineering (TSE)*, SE-5(2):264–277, 1979.
- [18] L. Bouge; and N. Francez. A Compositional Approach to Superimposition. *In Proceedings of the International Symposium on Principles of Programming Languages (POPL)*, pages 240–249. ACM Press, 1988.
- [19] D. J. Barnes and M. Kölling. Objects First with Java – A Practical Introduction using BlueJ. Prentice Hall / Pearson Education, 3rd edition, 2006.
- [20] A. Bergel, S. Ducasse, and O. Nierstrasz. Classbox/J: Controlling the Scope of Change in Java. *In Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 177–189. ACM Press, 2005.
- [21] C. Bockisch, M. Haupt, M. Mezini, and K. Ostermann. Virtual Machine Support for Dynamic Join Points. *In Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD)*, pages 83–92. ACM Press, 2004.
- [22] D. Batory, J. Liu, and J. N. Sarvela. Refinements and Multi-Dimensional Separation of Concerns. *In Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 48–57. ACM Press, 2003.
- [23] J. Hannemann, G. C. Murphy, and G. Kiczales. Role-Based Refactoring of Crosscutting Concerns. *In Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD)*, pages 135–146. ACM Press, 2005.
- [24] S. Hanenberg and A. Schmidmeier. Idioms for Building Software Frameworks in AspectJ. *In AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, 2003. Published at the workshop Web site: <http://www.cs.ubc.ca/~ycoady/acp4is03/papers.html>.

- [25] N. Ongkingco, P. Avgustinov, J. Tibble, L. Hendren, O. de Moor, and G. Sittampalam. Adding Open Modules to AspectJ. *In Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD)*, pages 39–50. ACM Press, 2006.
- [26] T. Reenskaug, E. Andersen, A. Berre, A. Hurlen, A. Landmark, O. Lehne, E. Nordhagen, E. Ness-Ulseth, G. Oftedal, A. Skaar, and P. Stenslet. OORASS: Seamless Support for the Creation and Maintenance of Object- Oriented Systems. *Journal of Object-Oriented Programming*, 5(6):27–41, 1992.
- [27] M. Sihman and S. Katz. Superimpositions and Aspect-Oriented Programming. *The Computer Journal*, 46(5):529–541, 2003.
- [28] R. Lincke, J. Lundberg and W. Löwe. “July. Comparing software metrics tools”. In *Proceedings of the 2008 International Symposium On Software Testing and Analysis*. ACM, pp. 131-142, 2008.
- [29] M. C. Lee. “Software quality factors and software quality metrics to enhance software quality assurance”. *British Journal of Applied Science and Technology*, vol. 4, no. 21, pp. 3069-3095, 2014.
- [30] S. Yeresime, J. Pati and S. K. Rath. “Review of software quality metrics for object-oriented methodology”. In *Proceedings of International Conference on Internet Computing and Information Communications*. Springer, India, pp. 267-278, 2014.
- [31] B. J. Eric and M. E. Bernstein. “*Software Engineering: Modern Approaches*”. Waveland Press, Long Grove, Illinois, USA, 2016.
- [32] M. I. Ghareb and G. Allen. April. “State of the art metrics for aspect oriented programming”. In *AIP Conference Proceedings*. Vol. 1952. AIP Publishing, p. 020107, 2018.
- [33] H. Noviyarto and Y. S. Sari. “Testing and implementation outpatient information system using ISO 9126”. *International Educational Journal of Science and Engineering*, vol. 2, no. 3, p. 11, 2019.
- [34] M. W. Suman and M. D. U. Rohtak. “A comparative study of software quality models”. *International Journal of Computer Science and Information Technologies*, vol. 5, no. 4, pp. 5634-5638, 2014.
- [35] G. Allen and M. Ghareb. Identifying similar pattern of potential aspect-oriented functionalities in software development life cycle”. *Journal of Theoretical and Applied Information Technology*, vol. 80, no. 3, pp. 491-499, 2015.

- [36] M. Ghareb and G. Allen. “*Improving the Design and Implementation of Software Systems uses Aspect-Oriented Programming*”. University of Human Development Sulimananay, Iraq, 2015.
- [37] H. Rashidi and M. S. Hemayati. “Software quality models: A comprehensive review and analysis”. *Journal of Electrical and Computer Engineering Innovations*, vol. 6, no. 1, pp. 59-76, 2019.
- [38] H. Kuwajima and F. Ishikawa. “*Adapting SQuaRE for Quality Assessment of Artificial Intelligence Systems*”. Machine Learning, arXiv preprint arXiv:1908.02134, 2019.
- [39] G. O’Regan. “Fundamentals of Software Quality. In *Concise Guide to Software Testing*. Springer, Cham, pp. 1-31, 2019.
- [40] ISO/IEC 9126 1, 2001, ISO/IEC 9126 2, 2003, ISO/IEC 9126 3, 2003 and ISO/IEC 9126 4. “*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, 2004.
- [41] Y. S. Sari. “Testing and implementation ISO 9126 for evaluation of prototype knowledge management system (KMS) e-procurement”. *International Educational Journal of Science and Engineering*, vol. 2, no. 3, p. 1, 2019.
- [42] A. Kaur, P. S. Grover and A. Dixit. “*Performance Efficiency Assessment for Software Systems. In Software Engineering*”. Springer, Singapore, pp. 83-92, 2019.
- [43] S. Arun, R. Kumar and P. S. Grover. “Estimation of quality for software components: An empirical approach”. *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 6, pp. 1-10, 2008
- [44] M. S. Deutsch and R. R. Wills. “*Software Quality Engineering; A Total Technical and Management Ap-proach*”. Prentic-Hall, Inc., Upper Saddle River, NJ, 1998.
- [45] Y. U. Mshelia and S. T. Apeh. “Can software metrics be unified”? In *International Conference on Computational Science and Its Applications*. Springer, Cham, pp. 329-339, 2019.
- [46] K. Sirbi and P. J. Kulkarni. “AOP and its impact on software quality”. *Elixir Computer Science and Engineering*, vol. 54, pp. 12606- 12610, 2013.
- [47] Berard, E. V. (1995). Metrics for object-oriented software engineering. <http://www.ipipan.gda.pl/~marek/objects/TOA/moose.html>. 2.2.3, 4.2, 5.8.
- [48] Whitmire, S. A. (1997). Object-Oriented Design Measurement. Wiley Computer Pub. 2.1, 2.2.3.

- [49] Manoj, H. M., & Nandakumar, A. N. (2016). Constructing Relationship between Software Metrics and Code Reusability in Object Oriented Design. *Aptikom Journal on Computer Science and Information Technologies*, 1(2), 63-76.
- [50] Chidamber, S. and Kemerer, C. A Metrics Suite for OO Design. *IEEE Trans. on Soft. Eng.*,20-6, June 1994, 476-493.
- [51] Mitchell Wand and GregorKiczales, "A Semantics for Advice and Dynamic Join Points in Aspect-Oriented Programming," *Transactions on Programming Languages and Systems TPLS*, vol. 26, 2004.
- [52] N. Mohammed, A. Govardhan, Comparison between Traditional Approach and Object-Oriented Approach in Software Engineering Development, *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 6, 2011.
- [53] V. R. Basili, L. Briand and W.L. Melo, "A Validation Of Object-Oriented Design Metrics As Quality Indicators", *Technical Report, Univ. of Maryland, Dep. of Computer Science, College Park, MD, 20742 USA*. April 1995.
- [54] P. Edith Linda, E. Chandra and J. Sharmila, "An Approach to Evaluate Object Oriented Class Structure using Score Carding Framework", *International Journal of Software Engineering and Its Applications*, vol. 9, No. 3, pp. 9-16, 2015.
- [55] Robillard, M. P. and Murphy, G. C. Static analysis to support the evolution of exception structure in object-oriented systems. *ACM Trans. Softw. Eng. Methodol.* 12, 2 (Apr. 2003), 191-221.
- [56] Bernardi, M. L., & di Lucca, G. A. (2007). An interprocedural aspect control flow graph to support the maintenance of aspect oriented systems.*IEEE International Conference on Software Maintenance (ICSM2007)*, 2-5 Oct, 435–444.
- [57] Zhao, J. (2002). Slicing aspect-oriented software. *Proceedings of 10th International Workshop on Program Comprehension*, 251–260.
- [58] IEEE Std 1219-1998. (1998). *IEEE Standard for Software Maintenance*. Software Engineering Standard, IEEE Computer Society.
- [59] Lientz, B, & Swanson, E. (1980). *Software maintenance management*. Reading, Mass.: Addison-Wesley,(p.214).
- [60] Arora, V., Bhatia, R. K., & Singh, M. (2012). Evaluation of flow graph anddependence graphs for program representation. *International Journal of Computer Applications*, 56(14):18–23.

- [61] Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using Program Slicing To Improve the Efficiency and Effectiveness of Cluster Test Selection, *International Journal of Software Engineering and Knowledge Engineering*, 21(06): 759–777.
- [62] Gold, R. (2010). Control flow graphs and code coverage. *International Journal of Applied Mathematics and Computer Science*, 20(4):739–749.
- [63] Gold, R., (2014), Reductions of Control Flow Graphs. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 8(3), 143 427–434. Retrieved from <http://waset.org/publications/9997636/reductions-ofcontrol-flow-graphs>.
- [64] Moreira, A., Chitchyan, R., Araujo, J., & Rashid, A. (eds.) (2013). *Aspect-oriented Requirements Engineering*. New York: Springer.
- [65] Lippert, M and Lopes, C. A Study on Exception Detection and Handling Using Aspect-Oriented Programming. *In Proc. of the 22nd ICSE*, pages 418-427, Limerick, Ireland, 2000.
- [66] Baker, A. L., Bieman, J. M., Fenton, N., Gustafson, D. A., Melton, A., and Whitty, R. (1990). A philosophy for software measurement. *J. Syst. Softw.*, 12(3):277–281.
- [67] Card, D. N. and Glass, R. L. (1990). *Measuring Software Design Quality*. Prentice-Hall.
- [68] Armour, P. G. (2004). Beware of counting loc. *Commun. ACM*, 47(3):21–24.
- [69] Sears, A. (1993). Layout appropriateness: A metric for evaluating user interface widget layout. *IEEE Trans. Softw. Eng.*, 19(7):707–719.
- [70] Paulish, D. J. and Carleton, A. D. (1994). *Case studies of software-process-improvement measurement. Computer*, 27(9):50–57.
- [71] Lorenz, M. and Kidd, J. (1994). *Object-oriented Software Metrics:A Practical Guide. PTR Prentice Hall*.
- [72] Chidamber, S. R. and Kemerer, C. F. (1991). Towards a metrics suite for object oriented design. *In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications*, pages 197–211, New York, NY, USA. ACM Press.
- [73] Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493.
- [74] Weller, E. F. (1994). Using metrics to manage software projects. *Computer*, 27(9):27–33.
- [75] IEEE (1994). *Software Engineering Standards, 1994 edition. IEEE*.
- [76] Binder, R. V. (1994). Object-oriented software testing. *Commun. ACM*,37(9):28–29.
- [77] Humphrey, W. S. (2005). *PSP: A Self-Improvement Process for Software Engineers*. Addison-Wesley.

- [78] Humphrey, W. S. (2006). *TSP: Leading a Development Team*. Addison-Wesley.
- [79] Harrison, R., Counsell, S. J., and Nithi, R. V. (1998). An evaluation of the mood set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, 24(6):491–496.
- [80] Whitmire, S. A. (1997). *Object-Oriented Design Measurement*. Wiley Computer Pub.
- [81] Demeyer, S., Ducasse, S., and Nierstrasz, O. (2000). Finding refactorings via change metrics. *SIGPLAN Not.*, 35(10):166–177.
- [82] Pressman, R. S. (2000). *Software Engineering: A Practitioners Approach*. McGraw Hill.
- [83] Sotirovski, D. (2001). Heuristics for iterative software development. *IEEE Software*, 18(3):66–73.
- [84] Lanza, M. (2001). The evolution matrix: recovering software evolution using software visualization techniques. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 37–42, New York, NY, USA. ACM Press.
- [85] Kabaili, H., Keller, R. K., and Lustman, F. (2001). Cohesion as changeability indicator in object-oriented systems. In *CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, page 39, Washington, DC, USA. IEEE Computer Society.
- [86] Mens, T. and Demeyer, S. (2001). Future trends in software evolution metrics. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 83–86, New York, NY, USA. ACM Press.
- [87] Ramil, J. F. and Lehman, M. M. (2001). Defining and applying metrics in the context of continuing software evolution. In *METRICS '01: Proceedings of the 7th International Symposium on Software Metrics*, page 199, Washington, DC, USA. IEEE Computer Society.
- [88] Rifkin, S. (2001). What makes measuring software so hard? *IEEE Softw.*, 18(3):41–45.
- [89] Buglione, L. and Abran, A. (2001). Creativity and innovation in spi: an exploratory paper on their measurement? In *IWSM'01: International Workshop on Software Measurement*, pages 85–92, Montreal, Quebec, Canada.
- [90] C. Zhang, D. Gao, and H.-A. Jacobsen. Towards Just-In-Time Middleware Architectures. In *Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD)*, pages 63–74. ACM Press, 2005.
- [91] S. Trujillo, D. Batory, and O. Diaz. Feature Refactoring a Multi-Representation Program into a Product Line. In *Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE)*, pages 191–200. ACM Press, 2006.

- [92] Manuel Wimmer et al., "A Survey on UML-Based Aspect-Oriented Design Modeling," *ACM Computing Surveys*, vol. 43, no. 4, 2011.
- [93] Peng Wang and Xiaochun Zhao, "The Research of Automated Select Test Cases for Aspect-oriented Software," *Information Engineering Research Institute*, 2012.
- [94] Elisa Y Nakagawa, Fabiano C Ferrari, Mariela M.F Sasaki, and José C Maldonado, "An Aspect-Oriented Reference Architecture for Software Engineering Environments," *The Journal of Systems and Software*, vol. 84, 2011.
- [95] Booch, G. (2008). Tribal memory. *IEEE Software*, 25(2):16–17.
- [96] Cleland-Huang, J., Chang, C. K., and Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Trans. Softw. Eng.*, 29(9):796–810.
- [97] D. Wu, L.Chen, Y. Zhou and B. Xu, "A metrics-based comparative study on object-oriented programming languages", *State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China*, DOI reference number: 10.18293/SEKE2015-064, 2015.
- [98] P. Edith Linda, E. Chandra and J. Sharmila, "An Approach to Evaluate Object Oriented Class Structure using Score Carding Framework", *International Journal of Software Engineering and Its Applications*, vol. 9, No. 3, pp. 9-16, 2015.
- [99] Zavvar, M., Garavand, S., Nehi, M.R., Yanpi, A., Rezaei, M. and Zavvar, M.H., 2017. Measuring Reliability of Aspect-Oriented Software Using a Combination of Artificial Neural Network and Imperialist Competitive Algorithm. *Asia-Pacific Journal of Information Technology and Multimedia*, 5(2).
- [100] Kaur, Mandeep, and Rupinder Kaur. "Improving the design of Cohesion and coupling metrics for aspect oriented software development." *Int. J. Comput. Sci. Mob. Comput*4, no. 5 (2015): 99-106.
- [101] Alemneh, E., 2014. Current States of Aspect Oriented Programming Metrics. *International Journal of Science and Research*, 3(1), pp.142-146.
- [102] Ghareb, M.I. and Allen, G., 2015. Identifying Similar Pattern of Potential Aspect Oriented Functionalities in Software Development Life Cycle. *Journal of Theoretical and Applied Information Technology*, 80(3), p.491.
- [103] Arora, K., Singhal, A. and Kumar, A., 2012. A Study of Cohesion Metrics for Aspect Oriented Systems. *Int. J. Eng. Sci. Adv. Tech*, 2(2), pp.332-337.
- [104] AnkushVesra, Rahul "A Study of Various Static and Dynamic Metrics for Open Source Software", *International Journal of Computer Applications* (0975 – 8887) Volume 122 – No.10, July 2015.



- [105] 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.
- [106] 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.
- [107] K. Mguni and Y. Ayalew, “Improving maintainability in COTS based system using aspect oriented programming: an empirical evaluation,” in *Proceedings of African Conference of Software Engineering and Applied Computing*, pp. 21–28, IEEE Computer Society, Gaborone, Botswana, 2012.
- [108] Johannes Braver, Reinhold Plosch, Matthias Saft and Christian Korner, “A Survey on the Importance of Object-Oriented Design Best Practices”, *43rd Euromicro Conference on Software Engineering and Advanced Applications, IEEE*, 2017.
- [109] Gulia, P., Khari, M., and Patel, S. (2019). Metrics analysis in object oriented and aspect oriented programming. *Recent Patents on Engineering*, 13(2), 117–122.
- [110] Hentunen, D. (2015). Detecting a return-oriented programming exploit. Google Patents.
- [111] Hassan, B. A. (2020). *CSCF: a chaotic sine cosine firefly algorithm for practical application problems. Neural Computing and Applications*, 120.
- [112] Hassan, B. A. (2021). Analysis for the overwhelming success of the web compared to microcosm and hyper-G systems. ArXiv Preprint ArXiv:2105.08057.
- [113] Hassan, B. A., Ahmed, A. M., Saeed, S. A., and Saeed, A. A. (2016). Evaluating e-Government Services in Kurdistan Institution for Strategic Studies and Scientific Research Using the EGOVSAT Model. *Kurdistan Journal of Applied Research*, 1(2), 1–7.
- [114] Hassan, B. A. and Qader, S. M. (2021). A New Framework to Adopt Multidimensional Databases for Organizational Information System Strategies. ArXiv Preprint ArXiv:2105.08131.
- [115] Hassan, B. A. and Rashid, T. A. (2019). Operational framework for recent advances in backtracking search optimisation algorithm: A systematic review and performance evaluation. *Applied Mathematics and Computation*, 124919.
- [116] Hassan, B. A. and Rashid, T. A. (2021a). A multidisciplinary ensemble algorithm for clustering heterogeneous datasets. *Neural Computing and Applications*. URL: <https://doi.org/10.1007/s00521-020-05649-1>

- [117] Hassan, B. A. and Rashid, T. A. (2021b). Artificial Intelligence Algorithms for Natural Language Processing and the Semantic Web Ontology Learning. ArXiv Preprint ArXiv:2108.13772.
- [118] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. (2001). Getting started with AspectJ. *Communications of the ACM*, 44(10), 59–65.
- [119] Lau, R. Y. K., Zhang, W., and Xu, W. (2018). Parallel aspect-oriented sentiment analysis for sales forecasting with big data. *Production and Operations Management*, 27(10), 1775–1794.
- [120] Lemos, O. A. L., Ferrari, F. C., Masiero, P. C., and Lopes, C. V. (2006). Testing aspect-oriented programming pointcut descriptors. *Proceedings of the 2nd Workshop on Testing Aspect-Oriented Programs*, 33–38.
- [121] Mcheick, H. and Godmaire, S. (2018). Designing and implementing different use cases of aspect-oriented programming with AspectJ for developing mobile applications. *Proceedings of the 7th International Conference on Software Engineering and New Technologies*, 1–8.
- [122] Maarooof, B. B., Rashid, T. A., Abdulla, J. M., Hassan, B. A., Alsadoon, A., Mohamadi, M., ... Mirjalili, S. (2022). Current Studies and Applications of Shuffled Frog Leaping Algorithm: A Review. *Archives of Computational Methods in Engineering*, 1–16.
- [123] Nguyen, T. (2018). Java Spring Framework in developing the Knowledge Article Management application: A brief guide to use Spring Framework.
- [124] Panwar, P., Agarwal, D., Vyas, P., Jadhav, P. A., and Joshi, S. D. (2019). Evolution of Testing with Respect to the Programming Paradigms. *International Journal of Mechanical Engineering and Technology*, 10(3).
- [125] Rademacher, F., Sachweh, S., and Zündorf, A. (2019). Aspect-oriented modeling of technology heterogeneity in microservice architecture. *2019 IEEE International Conference on Software Architecture (ICSA)*, 21–30.
- [126] Raheman, S. R., Maringanti, H. B., and Rath, A. K. (2018). Aspect oriented programs: Issues and perspective. *Journal of Electrical Systems and Information Technology*, 5(3), 562–575.
- [127] Rajan, H. and Sullivan, K. J. (2005). Classpects: unifying aspect-and object-oriented language design. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 59–68.

- [128] Saeed, M. H. R., Hassan, B. A., and Qader, S. M. (2017). An Optimized Framework to Adopt Computer Laboratory Administrations for Operating System and Application Installations. *Kurdistan Journal of Applied Research*, 2(3), 92–97.
- [129] Vidal, C., Benavides Cuevas, D. F., Leger, P., Galindo Duarte, J. Á., and Fukuda, H. (2015). Mixing of join point interfaces and feature-oriented programming for modular software product line. *BICT 2015: 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (2015)*, p 433-437.
- [130] Wand, M., Kiczales, G., and Dutchyn, C. (2001). A semantics for advice and dynamic join points in aspect-oriented programming. *SAIG*, 45–46.
- [131] Zhang, L. (2011). Study on comparison of AOP and OOP. *2011 International Conference on Computer Science and Service System (CSSS)*, 3596–3599.

## LIST OF PUBLICATIONS:

### -CONFERENCES-

1. Shrikant Patel, KavitaSaini, "Performances of Various Aspect-Oriented Systems with Their Significant Quality Attributes" in 6<sup>th</sup> International Conference on Contemporary Computing and Informatics -Amity University. (Scopus)
2. Shrikant Patel, Kavita Saini, Md. Noman Khan, Chaman Kumar, " Evaluation of Tracking System using Facial Recognition and Location" in 5<sup>th</sup> IEEE International Conference on Advances in Computing, Communication Control and Networking (ICAC3N-23)-Galgotias College of Engineering. (Scopus)
3. Patel S., Sharma N., Kumar S., "An Approach To Reduce Error in Regression Problems with an Implementation" 2022<sup>9<sup>TH</sup></sup> *International Conference on Computing for Sustainable Global Development (INDIACom)*, 23-25 March-2022, pp. 119-125. **Electronic ISBN:978-93-80544-44-1.**(Scopus)
4. S. Patel, V. Raj and M. Harit, "Optimization of Storage Management in Android," 2018 5<sup>th</sup> *International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE Conference ID-42835, 14<sup>th</sup> to 16<sup>th</sup> March 2018, ISSN 0973-7529; ISBN 978-93-80544-28-1, PP-5032-5035.
5. S. Patel, S. Kumar, S. Katiyar and R. Chaudhary- "MONGODB VSMYSQL: A Comparative Study of MongoDB and MySQL Based on Their Performance" in International Conference RICE 2020, ISBN- 978981157566 on 05 January 2020, DOI: 10.1007/978-981-15-7527-3\_6.(Springer)
6. S. Patel, A. Dev and P. Gulia, "Comparative analysis of object-oriented programming and aspect-oriented programming approach," 2015 *2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 04 May, 2015, pp. 1836-1842. **Electronic ISBN:978-9-3805-4416- 8, Print ISBN:978-9-3805-4415-1, CD:978-9-3805-4414-4.**(Scopus)

## **-JOURNALS-**

1. Shrikant Patel, Sanjay Kumar and Naveen Sharma- “Metric Analysis for AOP and OOP programming Paradigm” in Journal of The Institution of Engineers (India): Series B, 2023, 104(1), pp. 215–220.(Scopus)
2. Shrikant Patel, Preeti Gulia and Manju Khari- “Metrics Analysis in Object Oriented and Aspect Oriented Programming” in Recent Patents on Engineering (International Journal) at Bentham Science Publications on 27 May, 2019, Vol. 13, Issue 2, Page: [117 - 122], DOI-10.2174/1872212112666180831115458, Print ISSN-1872-2121, Online ISSN-2212-4047. (Scopus)
3. S. Patel, A. Sharma, S. Kumar, P. Lathar, “Rough Set Generalization Models Based Information Retrieval ” in International Journal of Health Sciences, 6(S5), 8660–8672, e-ISSN 2550-696X, ISSN-2550-6978.(Scopus)
4. S. Patel, “Code Reusability with aspect using object-oriented programming language”, in Journal of Harmonized Research in applied science, Volume-5, Issue-4 in October 2017, and ISSN-2321-7456. (UGC RECOGNIZED JOURNAL)
5. S. Patel, S.R. Jena, A.K. Yadav, M.H. Saibaba- “ANALYSIS ON MOBILE CLOUD SECURITY AND COMPARISON OF EXISTING MODELS” in Volume 12, No. 3, Page-580-590, e-ISSN-0976-5166 in May-June 2021. (Scopus)
6. Shrikant Patel, “HIGH PERFORMANCE ALGORITHM TO SOLVE RUBIK’S CUBE PROBLEM”, in Jai MaaSaraswatiGyandayini, Volume 3, Issue -2, October 2-17, ISSN 2454-8367.(UGC RECOGNIZED JOURNAL)
7. S. Patel, S. Kumar, S. Katiyar and S. Upadhyay- “Metrics Analysis in Aspect Oriented and Object-Oriented Programming based on Exception Handling” in International Journal of Advanced Science and Technology (IJAST) [International Journal], Vol. 29, No. 9s, (2020), pp. 8027-8038. (Scopus)
8. S. Patel, A. Tommy and S. Kumar- “The Impact of Covid-19 Pandemic on Global Education” in International Journal of Solid-State Technology (International Journal), ISSN-0038111X, Volume -63, Issue-2, December-2020. (UGC Recognized Journal)
9. Shrikant Patel, Sanjay Kumar and Nishu Sharma- “Software quality assurance and its requirements” in International Journal of Solid-State Technology, ISSN-0038111X, Volume -63, Issue-2, December-2020.(UGC Recognized Journal)
10. S. Patel, I. Jolly, G. K. Sharma and S. Kumar- “Blockchain with Artificial Intelligence” in IPEM JOURNAL OF COMPUTER APPLICATION AND RESEARCH (National Journal) in Vol. 5, pp-53-63, ISBN-2581-5571 on December 2020.

## **-PATENTS-**

**1. Application Number: 2021104788**

Filing Date 2021-08-01

Title:

ELECTRIC CAR DESIGN USING AI POWERED BATTERY TO ENHANCE THE BATTERY LIFETIME

Applicant(s):

Balusamy, Balamurugan; Dhanaraj, **Patel, Shrikant**; Rajesh Kumar; Haidri, Raza Abbas; Jena, Soumya Ranjan; Krishnasamy, Lalitha; Panigrahi, Rakhee; Patjoshi, Rajesh Kumar; Sabharwal, Munish

**2. Application Number: 2021104684**

Filing Date 2021-07-29

Title:

UNIQUE EV CAR DESIGN WITH DIVERSE SOLAR PANELS AND NO POWER DISPLAYS

Applicant(s):

Badal, Sandeep MR; Dhanaraj, Rajesh Kumar DR; **Patel, Shrikant MR**; Jena, Soumya Ranjan MR; Krishnasamy, Lalitha DR; Panigrahi, Rakhee DR; Patjoshi, Rajesh Kumar

**3. Application Number-202341018507**

Filing Date: 18-03-2023

Publication Date: 31-03-2023

Title:

A METHOD FOR SUSTAINABLE GREEN ENERGY OPTIMIZATION FOR EDGE CLOUD COMPUTING WITH RENEWABLE ENERGY RESOURCES

Applicant(s):

S.R. Jena, S.N. Manoharan, J.P. Singh, Harmandeep Singh, Ch. M.H. Saibaba, **Shrikant Patel**

**4. Application Number-202311022214**

Filing Date: 27-03-2023

Date of Publishing: 19-05-2023

Title:

IOT BASED SOLAR ASSISTANT AUTOMATIC SMART CHULLAH

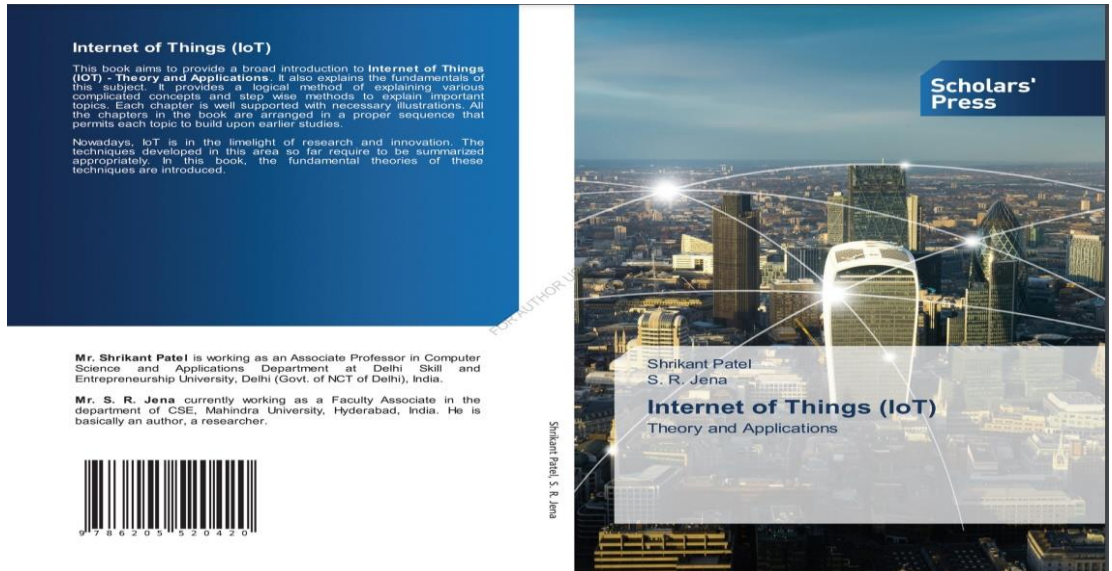
Applicant(s):

J.N. Singh, Sandeep Srivastava, Kavita Saini, Dr. Rajeev Kumar Sharma, Sonia Kukreja, Shrikant Patel, Rani Kumari, Yogita Chauhan, Dr. Pooja Saigal,

**-BOOKS -**

**1. Book Name- Internet of Things (IoT):Theory and Applications**

**Author Name- Shrikant Patel and S.R. Jena**



**2. Book Name- Research Methodology:Theory and Applications**

**Author Name- Shrikant Patel, S.R. Jena, Dr. Pankaj Lathar**

