Institute of Software Technology
University of Stuttgart
Universitätstraße 38
D-70569 Stuttgart

Comparing Service Orientation and Object Orientation: A Case Study on Structural Benefits and Maintainability

Author:	Bhupendra Choudhary
Course of Study:	Computer Science
Date of work begin:	01.12.2017
Date of submission:	01.06.2018
Supervisor:	M.Sc. Justus Bogner
Examiner:	Prof. Dr. rer. nat. Stefan Wagner

Service Orientation (SO) is a dominating technique evolving around the use of Object Orientation (OO). The conceptual comparison of both the approaches have been broadly explained in the literature, but the generalizable comparison of the maintainability of two paradigms is still a topic under research. This thesis tries to provide a generalized comparison of the maintainability using two functionally equivalent Online BookStore systems developed with Service Orientation and Object Orientation. This thesis presents a brief explanation of the software metrics used for the comparison. The quantitative comparison revealed that the Service-Oriented version of the system has a lower coupling and higher cohesion between software modules compared to an Object-Oriented approach. Through survey results, it was found that Service Orientation has a better degree of modifiability, encapsulation and abstraction while Object-Orientation provides a reduced degree of testing and system complexity comparatively. Also in expert interviews, participants believe that systems based on service orientation possess a better degree of stability, analyzability and modifiability whereas Object-Oriented System tends to provide a lower degree of structural complexity. Furthermore, experimental results suggest that a Service-Based System has a better degree of extensibility and changeability compared to Object-Oriented System.
Keywords: Service-Oriented Architecture(SOA), Object-Oriented Architecture, Internal Software Quality Attributes, Software Maintainability, Software Quality Metrics.

Contents

1 Introduction

1.1 Motivation

You can't manage what you can't control, and you can't control what you don't measure. To be effective software engineers or software managers, we must be able to control software development practice. If we don't measure it, however, we will never have that control.

- Tom DeMarco, Controlling Software Projects[1]

In the area of software engineering, developing a software system to satisfy its functional specification is important. Minimizing the cost and efforts that are required to develop a software system also takes similar importance. For this reason, software maintainability has been a challenging factor for every organization.
Many publications describe the significance of this factor and address the necessity of good maintainability in software systems. The majority of this researches is based on the principle that software systems with improved maintainability are prone to fewer defects[2] and result in cost savings. As acknowledged by Bass et al.[3], re-configuration of software systems is often not initiated because of insufficient functionality but due to significant maintainability-related problems.

Object Orientation (OO) has been a primary approach for developing software. It brought significant improvement in software quality by exploiting various structural properties. These properties are predominantly coupling, cohesion, size and complexity. In the past, numerous studies have realized their profound effect on maintainability[4, 5, 6]. Thus, quantification has played a major role in predicting software systems maintainability.

The plethora of business requirements has led to the adoption of Service Orientation (SO) as a paradigm for developing highly distributed software systems. Service Orientation has increased the level of abstraction again and promised even greater benefits such as reusability, loose coupling, encapsulation and composability.

The motivation behind this thesis is to investigate if there is a way of generalizing maintainability of Service Orientation on the one hand and object orientation on the other hand. By comparing the two paradigms in this regard, valuable insights for their application can be generated.

Especially, since Service Orientation (SO) promised an enhanced degree of maintainability by fine abstraction, encapsulation, reusability, loose coupling etc. So the question is, if this is really quantifiable then what guideline have to be adhered in order to leverage these effects in Service Orientation (SO).

Some studies exist that try to provide empirical evidence for this, but it is very hard to compare systems based on Object-Orientation and Service Orientation. Moreover, there are a lot of metrics available for measuring coupling, cohesion and maintainability, but not all Object-Oriented or Service-Oriented metrics are applicable.

As a result, research has been centered on providing a set of mutually consistent metrics for quantifying the extent of maintainability, gaining empirical insights into Service Orientation benefits, investigating existing limitations from the studies that compare Service-Based Systems and Object-Oriented-systems. Furthermore survey, expert interviews and experimental work have been performed to gain insights into the software maintainability of two alternatives.

1.2 Thesis outline

The whole outline of the thesis can be seen in figure 1.1. The thesis begins with the chapter *Introduction* which highlights how a comparative analysis of maintainability of Service Orientation and Object Orientation can result in insights how these paradigms are best applied.

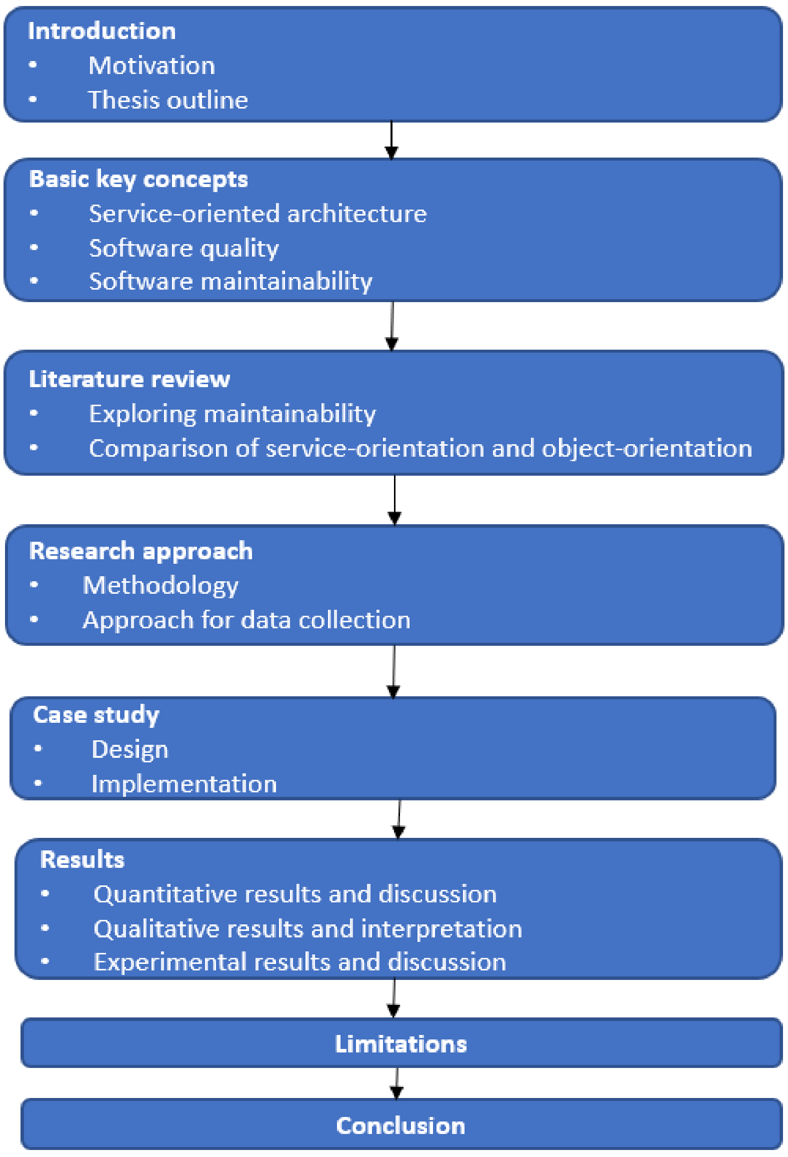

Introduction
Motivation
Thesis outline
Basic key concepts
Service-oriented architecture
Software quality
Software maintainability
Literature review
Exploring maintainability
Comparison of service-orientation and object-orientation
Research approach
Methodology
Approach for data collection
Case study
Design
Implementation
Results
Quantitative results and discussion
Qualitative results and interpretation
Experimental results and discussion
Limitations
Conclusion

Figure 1.1: An outline describing the organisation of the thesis

The next chapter presents the fundamental concepts of Service-Oriented Systems. It begins with the questions behind the evolution of Service Orientation approach. Next, the potential benefits of Service-Oriented Architecture are introduced. Furthermore, we present communication within a Service-Oriented environment. The next section explains the various Service Orientation principles and characteristics.

Following this, software quality concepts are explored and we present various quality characteristics and approaches. The next section explains software maintainability where we mention the maintenance types and the factors that influence software maintainability. Also, we introduce various traditional metrics that are used to quantify the factors affecting maintainability.

The section thereafter describes the various maintainability affecting factors that are addressed in the literature. This chapter also explores the software metrics that are employed to quantify the factors that help to predict software maintainability. This section further presents the similarities and differences between Service Orientation and Object Orientation. This section ends with the investigation of the studies that compare Service Orientation and Object Orientation based on structural quality attributes.

The subsequent section explains the overall research methodology that has been executed in the thesis. It also includes the various approaches that have been applied to gain the insights into software maintainability.

The next chapter works around the design of the case study which is based on the design of quantitative, qualitative and experimental approach.

The results describes the data collected from quantitative, qualitative and experimental measures that are presented to satisfy our research objective.

In the end, a summary and future work are presented to concludes the thesis.

In a nutshell, different studies are explored that compares Service Orientation with Object Orientation. Also several metrics are investigated to measure the degree of coupling, cohesion and maintainability.

2 Key Concepts

2.1 Service-Oriented Architecture

While the IT industry has been continuously dealing with the challenges of minimizing costs and maximizing the exploitation of existing technologies, simultaneously they have to strive to cater clients effectively continually.

Thus business demands and growth has led to the broad acceptance of Service Orientation in the IT industry. It is continuously increasing company's agility over time. Service Orientation has been widely adoption in the software industry. It has increased company's agility[7] over time.

The Figure 2.1 represents the evolution of SOA.

There are two major factors behind all these exigencies: *Heterogenity* and *change*[8]. Today several organizations comprise heterogeneous architectures, applications, and systems which are too metaphorical and can be very complex and time-consuming. Integrating products from different suppliers were almost an agony. But the single-supplier approach to IT cannot be considered since supporting infrastructures and suite of applications are less resilient.

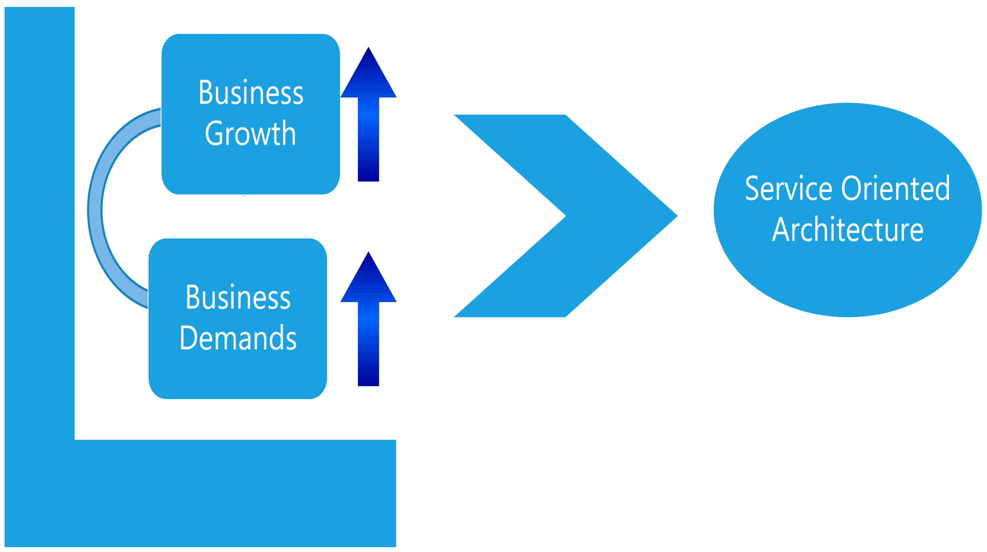


Figure 2.1: Evolution of Service-Oriented Architecture

Change is the second concern underlying the questions that IT-industries encounter. Continuously changing customer demands and requirements are driven by competitive markets readily available over the internet. In response, refinements in products and services expedite.

Technology continue to advance, nourishing increased client needs. Business must consolidate to stay in the competition, and IT infrastructure must empower businesses ability to customize.
Service Orientation is more of a characterization of IT evolution thus has a foundation in the past technologies and approaches.

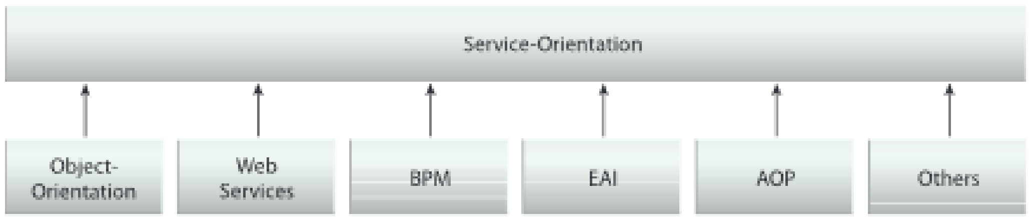


Figure 2.2: Primary influence of Service-Oriented Architecture [9]

As a result, some organizations are still moving towards service orientation, but a large percentage is arguably already using it. Based on Service-Oriented Architecture, service consumers do not have to worry about the services they are involved with because a service bus will select the right option acting on behalf of a service consumer. Underlying infrastructure tries to hide as many technical details as possible from the service requestor.

The below figure shows how the industry has benefitted from SOA.

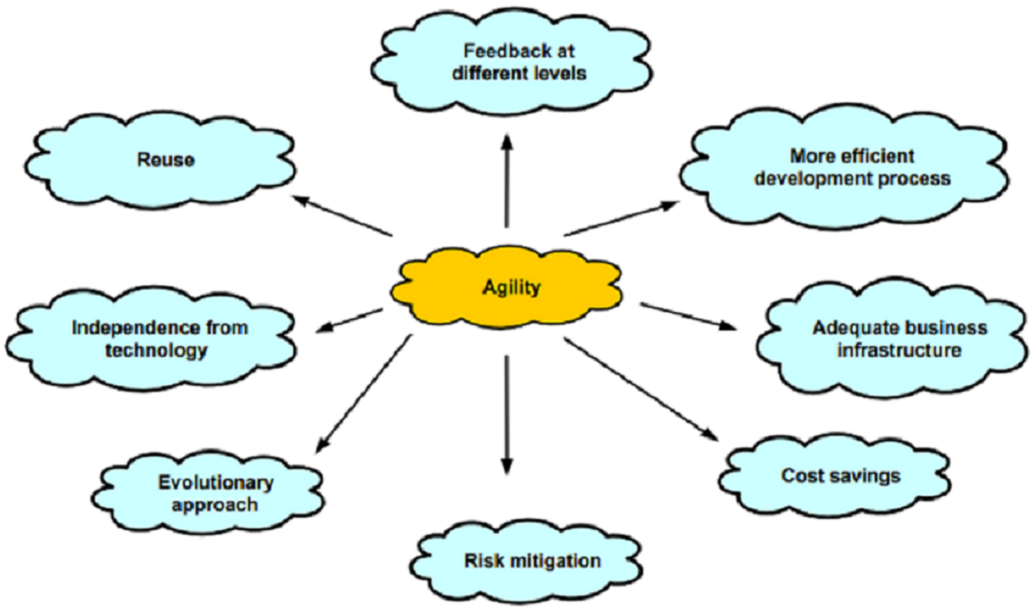

Feedback at different levels
Reuse
More efficient development process
Agility
Independence from technology
Adequate business infrastructure
Evolutionary approach
Cost savings
Risk mitigation

Figure 2.3: Potential benefits of Service-Oriented Architecture

The following figure depicts the basic Service-Oriented Architecture. It represents a service consumer sending service request to the service provider and in return service provider replies with a response message. Communication is established in such a way that is apparent to both the service consumer and service provider.

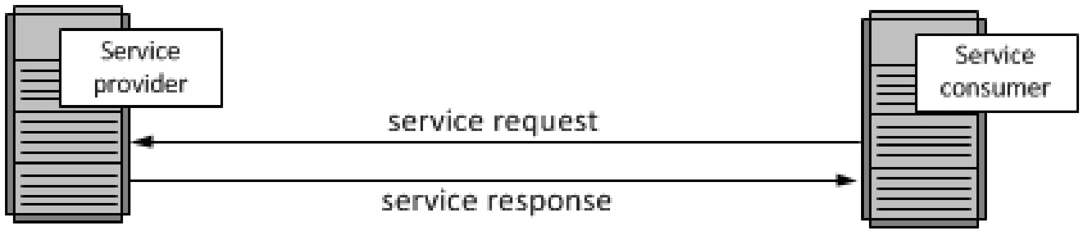


Figure 2.4: A basic Service-Oriented architecture[10]

Collaborations in SOA is based on publish, find, bind and invoke operations. Service and its description are the two artifacts that are contained by Service-Oriented architecture.

The diagram below shows the overall collaboration among the participants.

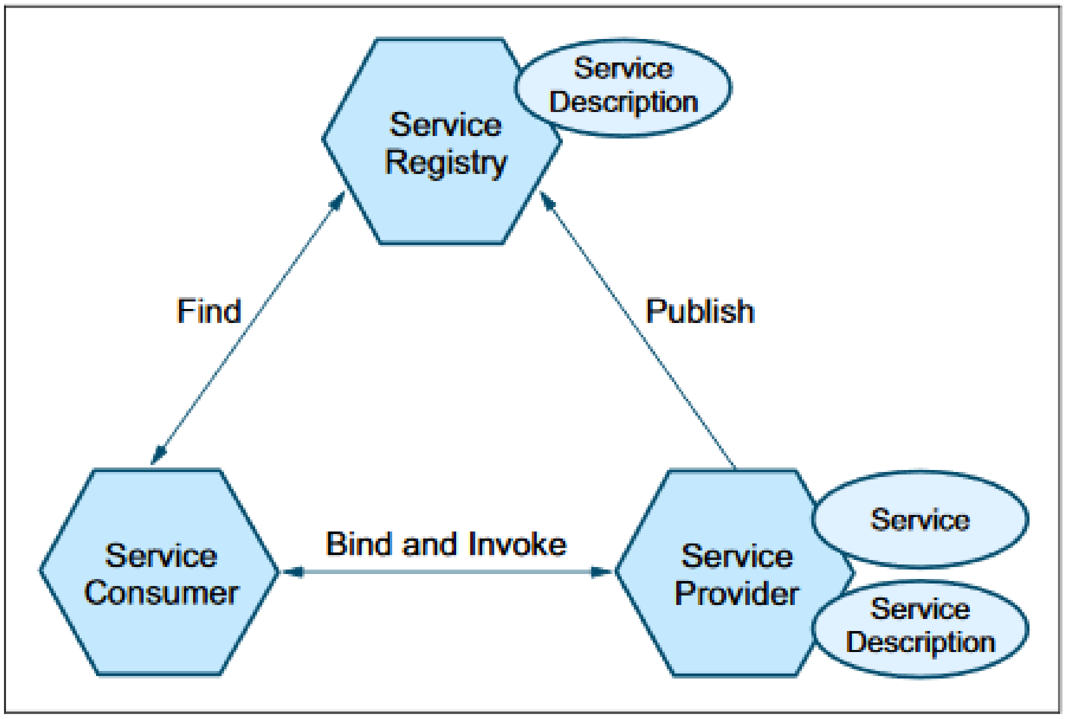


Figure 2.5: Collaborations in service-Oriented Architecture[11]

2.1.1 Principles of Service Orientation

The following section introduces you to the basic principles and characteristics that are attached to Service Orientation.
The figure below shows various design principles offered by Service Orientation.

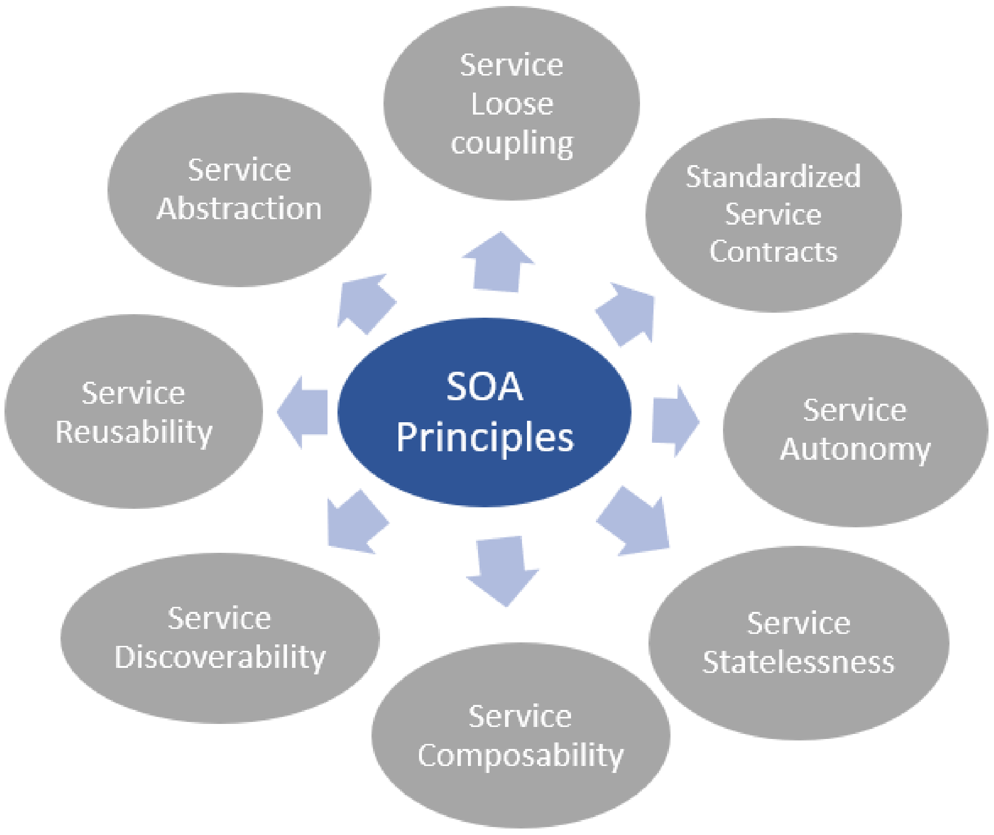


Figure 2.6: Principles of Service-Oriented Architecture[12]

Abstraction is considered as a core principle for any software development approach. Service Orientation has taken the concept of abstraction on a higher level. It has continued to evolve the higher-level abstraction to isolate change and to increase the granularity of reuse.
This principles enables the hiding of implementation level details for service consumers.

The figure 2.7 illustrates the need of abstraction in service environment.

It has been a challenge developing a system where consumers requests can be fulfilled over different channels without making any changes to the business logic each time when new channel support is expected.

This is where Service Orientation has played a significant role in dealing with such scenarios. Service abstraction layer employs an additional layer on the business level consisting all the necessary logic for devolving and receiving requests.

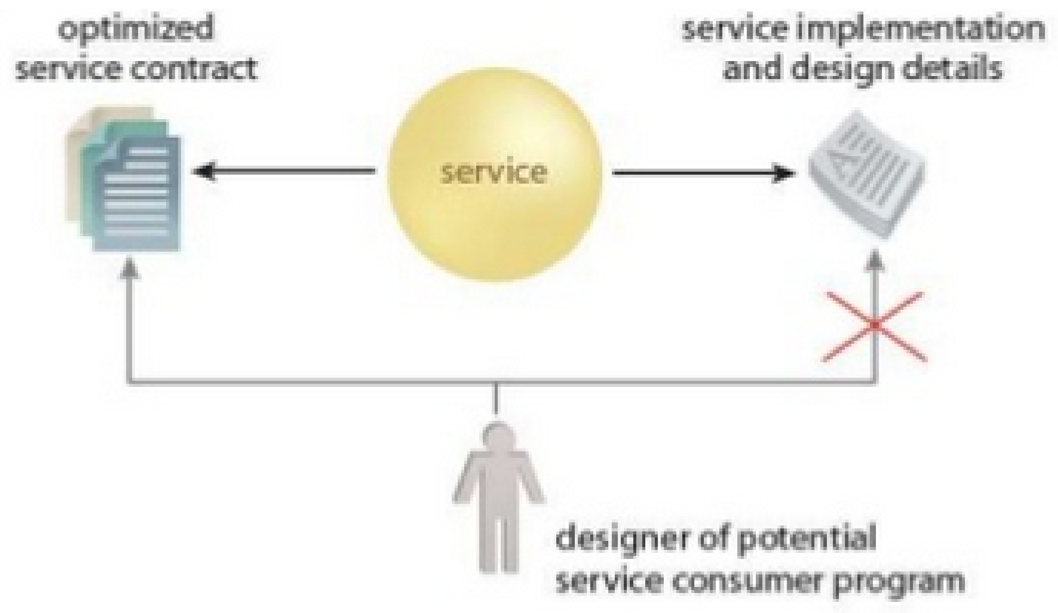

optimized service contract
service implementation and design details
service
designer of potential service consumer program

Figure 2.7: Why service abstraction needed [13]

Loose coupling is another essential aspect of Service Orientation. It has played a vital role in modernizing the legacy software. It has been improving business agility and making software systems to be easily scalable and guiding them towards an overall high-quality software.

The goal of decreasing/loosening coupling is to minimize the number and degree of dependencies in order to prevent ripple effects of changes. It has affected software quality directly. Tight coupling leads to the higher complexity in software systems which in turn makes the software difficult to understand, analyze and maintain. Loose coupling employs reusability so that software modules are developed once and can be used again and again. Also, it further increases the independence of software modules and enabled better flexibility.

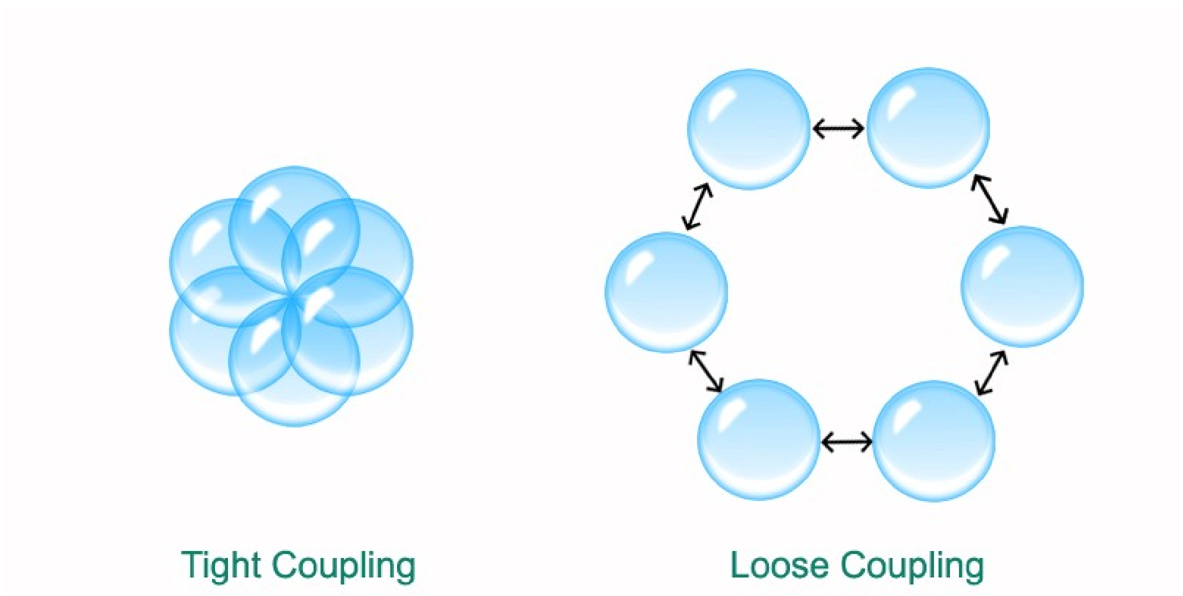


Figure 2.8: Coupling [14]

Reusability is a crucial factor in the maintainability of software systems and enhancing reusability is one of the fundamental objectives of Service Orientation; therefore, it is one of the highly advantages of SOA. Resultantly companies that have a restricted amount of success with reusability are considering SOA as an alternative solution.

The driving force behind service is that the logic at the back of service should not be restricted to a particular concern in an activity; it should cover multiple concerns in an activity related to the business processes. Thereby providing repeated values to the business.

Client application interacts with various services via a standard medium primarily internet. The same can be seen in the figure 2.9.
Well-designed SOA application permits infrastructure that promotes reusability in heterogeneous environments.

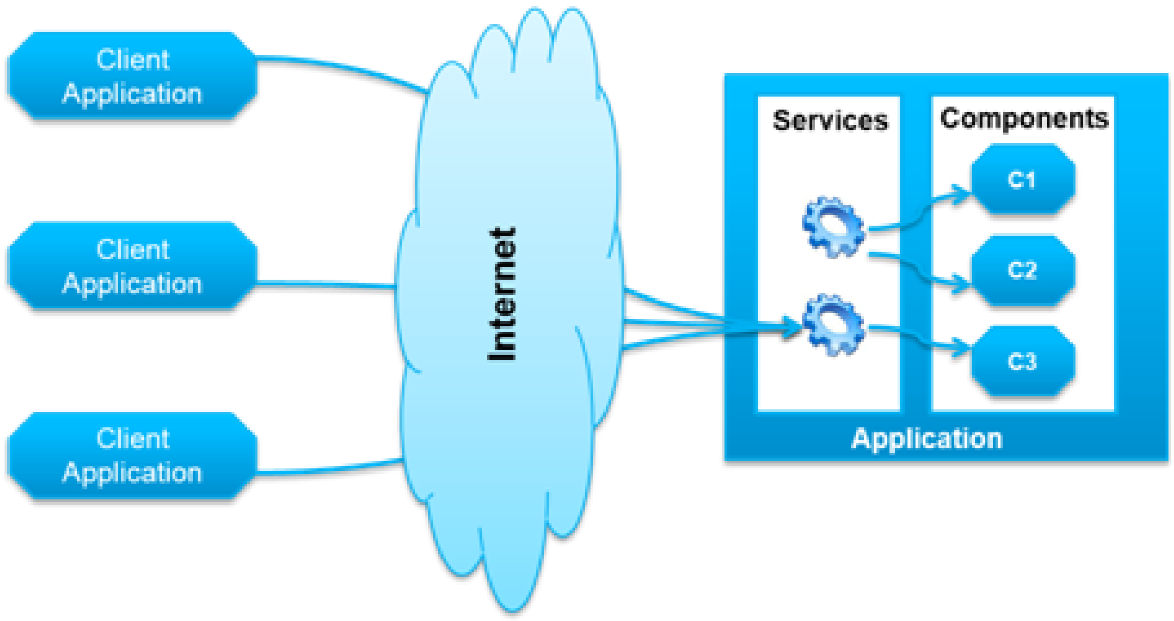


Figure 2.9: Reusability in Service Orientation [15]

"Services contain and express agnostic logic and can be positioned as reusable enterprise resources."[16] Thereby providing repeated values to the business.
Reuse is an advantageous part while inquiring SOA, but it is not the only primary advantage. Empowering organizations by bringing earlier detached systems can be highly effective.

Service reusability is correlated with the other design principles of SOA. The inter-relation among them can be seen in the figure 2.10.

Service reusability plays a key role in the composition of services. Service loose coupling minimizes the dependency among the services and enhances their reusability. Statelessness property of the services maximizes the reusability opportunities for services.
Service discovery provides a medium for the promotion of the reusability in services while service autonomy offers an execution environment for the service reuse.

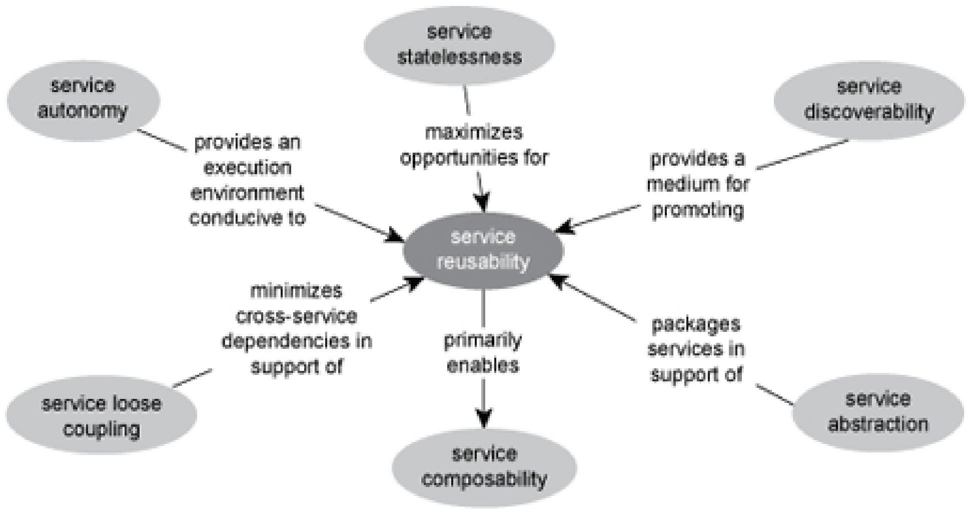

service statelessness
service autonomy
service discoverability
provides an execution environment conducive to
maximizes opportunities for
provides a medium for promoting
service reusability
minimizes cross-service dependencies in support of
primarily enables
packages services in support of
service loose coupling
service abstraction
service composability

Figure 2.10: Inter relation of service reusability with other design principles [17]

Service orientation is benefitted with another key principle that is known as service composability. This principle allows services to be designed in such a way that they can be reused extensively and enable organizations to offer rapid solutions to their customers.
A composite service is an extensive service made of several small services.

It directly impacts business agility by leveraging existing services. Implementation on consumer side become complex when services are consumed simultaneously by several service consumers. Buiding composite service makes the invocation simpler from the consumer perspective.

A service can be connected to several composite services; therefore, a single place of change enables them to manage comfortably and thus eventually provides a better degree of modifiability.

Also, impromptu solutions can be availed easily by utilizing the existing services. Already designed services can be merged to provide quick solutions. Thus new solutions can be employed cheaply by exploiting comparatively few services.

Furthermore unmasking few interfaces to the external consumers permits them to govern visibility.

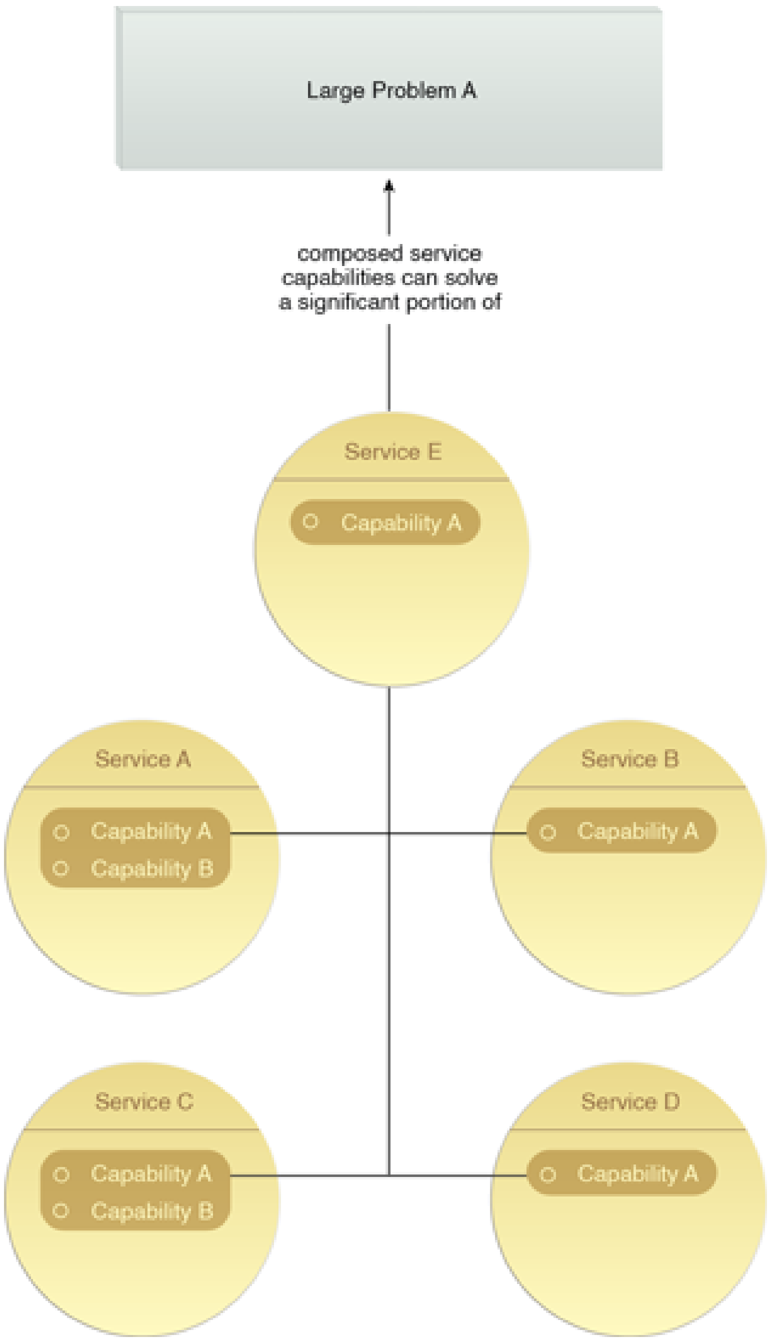


Figure 2.11: Service composition in Service Orientation [18]

2.1.2 Characteristics of Service Orientation

The figure 2.12 shows the primary characteristics offered by Service Orientation. Vendor-neutral solutions allow the underlying infrastructure readily adaptable to the frequently changing requirements. Not relying on a specific vendor makes an infrastructure effortlessly replaceable and gives them a chance to choose the best possible technology suited to their needs.

Business-driven characteristics of SOA need that the technology layer remains adaptable always. It means that the design of the underlying architecture continues to be flexible and scalable over a period.

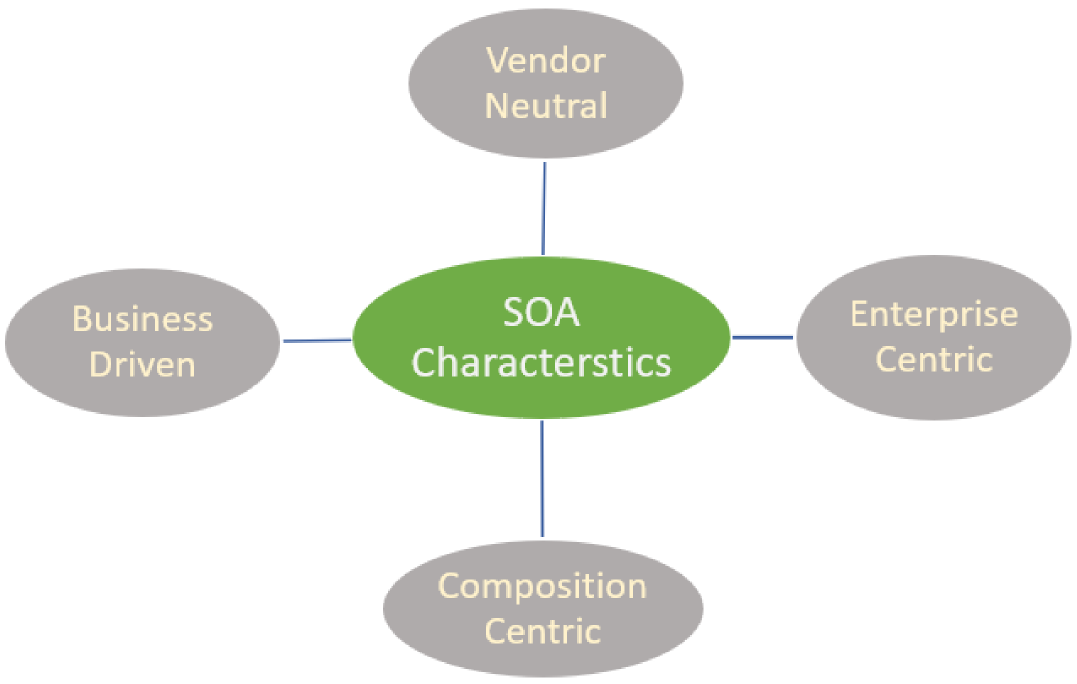


Figure 2.12: Characteristics of SOA[19]

Enterprise-centric property of SOA enforces that underlying technical layer foster services that are sparsed over traditional application silos. It is mainly concerned with shared services that be reused as a need for larger solutions [20].

As advocated in the service design principles, services are required to be composable so that they can repeatedly be used in the different solutions.

2.2 Software Quality

Quality defines software's success or failure in the competitive modern era. Therefore the task of maintaining quality is considered to be of great importance throughout the software development lifecycle.

Due to high importance, it has become the most looked after assets in the organizations. Industries have been continuously working on delivering better quality software.

Philip B. Crosby's "Quality is free: the art of making quality certain"[21] states:

The first erroneous assumption is that quality means goodness, or luxury or shininess.

The word "quality" is often used to signify the relative worth of something in such phrases as "good quality", "bad quality" and "quality of life" - which means different things to each and every person. As follows quality must be defined as "conformance to requirements" if we are to manage it. Consequently, the nonconformance detected is the absence of quality, quality problems become nonconformance problems, and quality becomes definable.

Quality attributes are dispersed among various layers of an application. Few of them contribute to the overall system design while some of them are limited to particular areas.

It is also essential to assess the effect of quality on the requirements while designing an application. The significance of the quality attributes varies from system to system.

ISO/IEC 1998 presented all the quality characteristics into further sub-characteristics. These sub-characteristics are shown in the figure below:

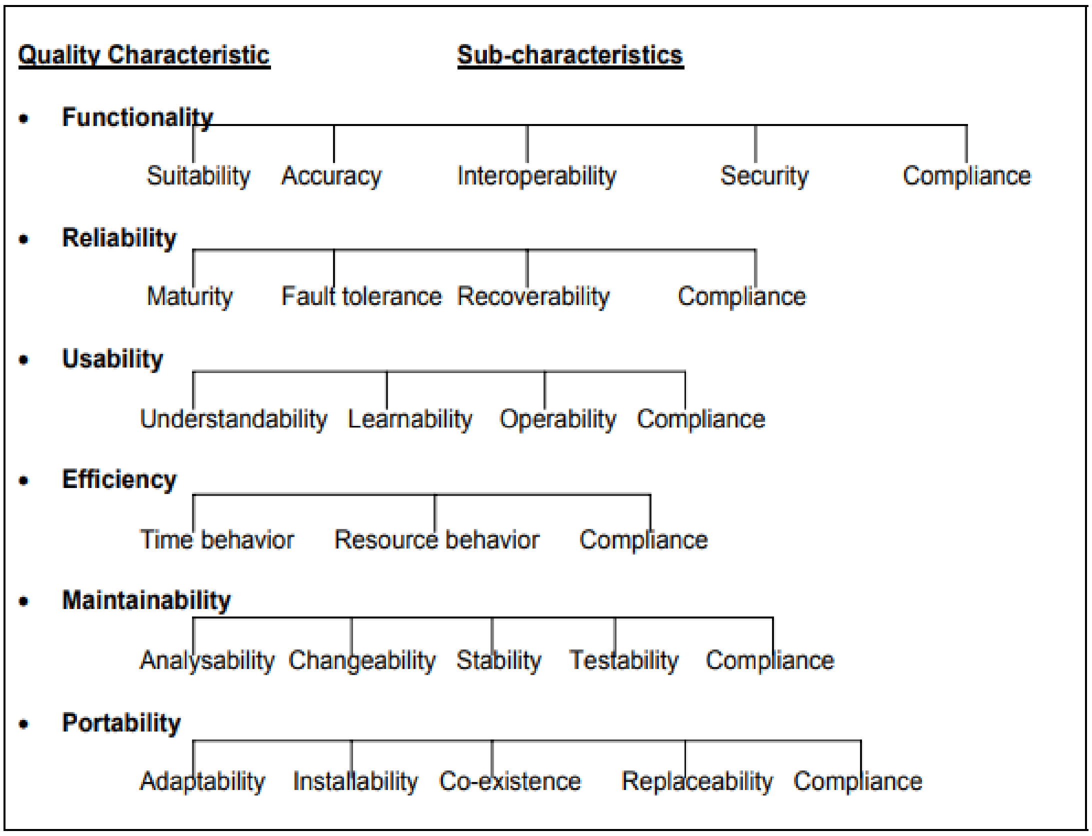


Figure 2.13: Software Quality Sub-characteristics [22]

In ISO 9126:1, three approaches have been defined for software quality: Internal quality, external quality and quality in use. These approaches affect each other as shown in the figure below.

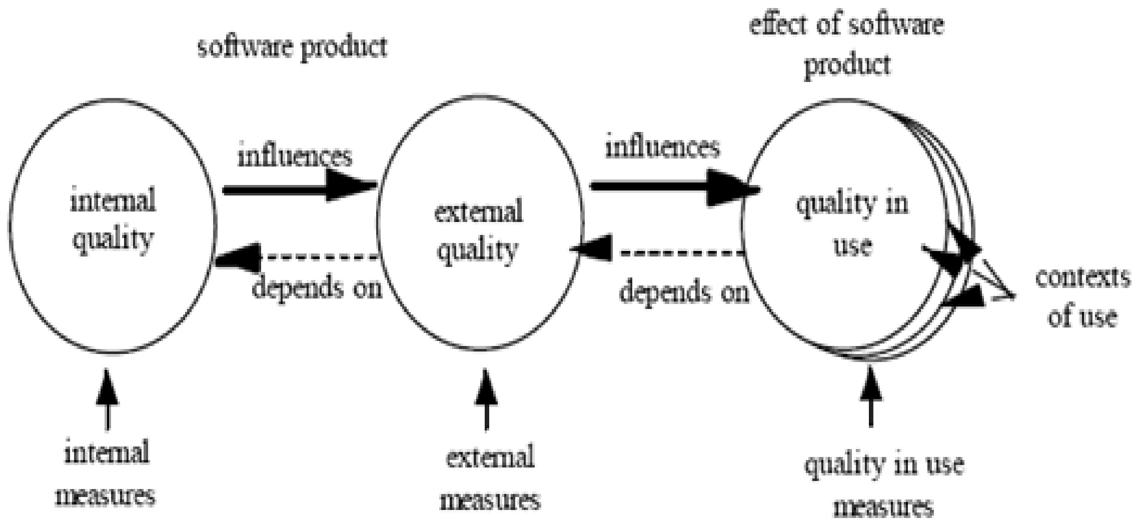


Figure 2.14: Various approaches to Quality [23]

Quality characteristics are used as a pillar to the internal and external quality of the software. These characteristics are refined until attributes are obtained. Then the software metrics are applied to measure these quality attributes. A relation between the elements is shown in the following figure.

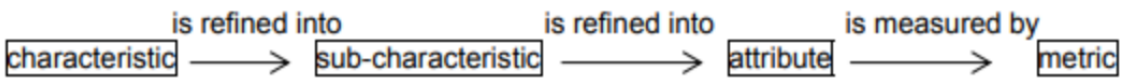


Figure 2.15: Relations among elements [22]

2.3 Software Maintainability

Software maintainability is defined[24] as *the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.*

Software maintainability consists of four types mentioned in the figure below.

Corrective maintenance is an action performed to rectify any bugs or errors occurred in a software system. It comes into place after a failure detection and aims at fixing a software system so that it can perform its desired functionality.

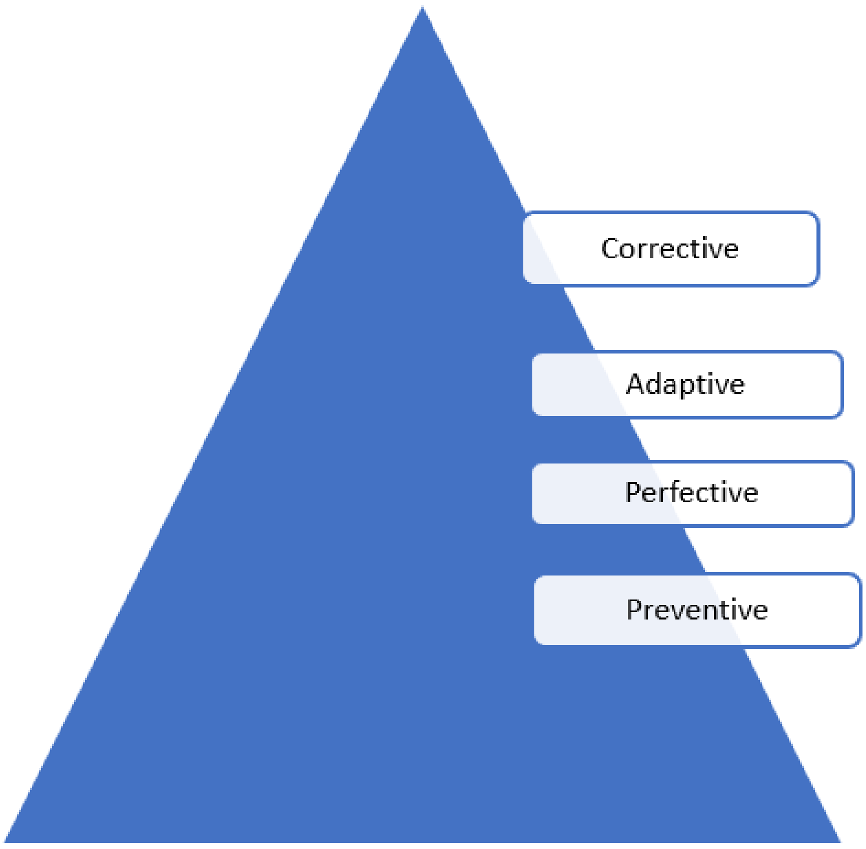


Figure 2.16: Types of software maintainability

While adaptive maintenance is focused on the modifications that are required to fulfill new user requirements. It can be initiated due to the variation in software, hardware, operating systems or in the business policies.

Perfective maintenance is triggered by the new or changed customer-requirements. It consists of making functional changes as well as the actions executed to improve systems performance. It also includes discarding the features that are no longer beneficial.

Preventive maintenance is also known as proactive maintenance. It focuses on enhancing software understandability and maintainability. Reorganization, code optimization, and system review are common activities in preventive maintenance.
Next section introduces to the factors that affect software maintainability.

2.3.1 Factors Affecting Maintainability

It is essential for the customers to measure the software maintainability. This section gives an overview of the factors that affect the maintenance of software systems.

As per ISO9126, maintainability is a key quality attribute in software systems. It includes four important attributes that influence maintainability. It consists of Analysability,modifiability, stability and testability represented in the table of figure below.

factors	**Description**
Analyzability	It involves the need to recognize errors are failure causes and to recognize the modifications of the parts of the software.
Modifiability	The steps taken to make changes in the system to make the implementation simpler.
Stability	It involves with the risk that come out of the results of unexpected modifications.
Testability	It validates the changes. It is the measurement of how the test criteria and the test components are done.

Table 2.1: Factors affecting code, documentation and tools [25]

David E. Peercy proposed a methodology for the evaluation of software maintainability. According to this methodology, software maintainability is majorly influenced by six key factors: descriptiveness, simplicity, instrumentation, modularity , consistency and expandability.

factors	**Description**
Modularity	It gives the description of the logical division of the software into numerous modules, parts and components. It is easy to understand and make changes in the software if it is composed of independent components.
Descriptiveness	It describes the explanations of the design. It gives the information about assumptions, components, objectives etc.
Consistency	It indicates the association of the products and the terminology, symbology and notations that are common.
Simplicity	There should be no complexity in the implementation techniques, languages used and the organization. It uses the concepts of the similarity.
Expandability	It describes that the changes in the physical information such as data storage, time of executions, computational procedures can be achieved easily.
Instrumentation	It helps in the improvement of testing.

Table 2.2: Factors affecting maintainability, based on methodology [26]

Researchers proposed a software maintainability model to enhance software quality. This model identifies several other factors that affect software maintainability.
These factors are described in the following table.

factors	**Description**
Readability	The extent of the code to which the user can understand.
Standardisation	The standards or guidelines which are available while writing a code.
Modularity	Decomposition of the system into numerous modules according to their functionalities to implement the abstractions of the data.
Programming Language	The code developed by a programmer in a language which can be reused.
Level of validation and testing	More time spending on validation and testing gives less errors and reduces the cost of maintenance.
Complexity	The difficulty in maintaining the code of the system should be put to a certain level.
Traceability	It traces the components to the requirements.

Table 2.3: Factors affecting maintainability, based on maintainability model [25]

2.3.2 Measuring Maintainability

The assessment of maintainability can help us to analyze the health of software systems. To assess maintainability, software metrics are applied. Therefore, software metrics play an important role in the evaluation of the maintainability.

There are many software metrics focused on quantifying software maintainability, but Chidamber and Kemerer metrics [27] are the most referenced in the literature. These metrics were designed explicitly for Object-Oriented Systems, but they are also used to measure the quality attributes affecting maintainability of Service-Oriented Systems.

The higher the count of *Number of Children* (NOC), the higher the level of reuse since inheritance is a form of reuse. Hence increasing the number of children improves the degree of reusability but it can also increase the testing complexities. *Weighted Methods Per Class* (WMC) count indicates the amount of time and efforts required to develop and maintain the class.
The increased size of the *Response For a Class* (RFC) count suggests a higher degree of testing complexities for a class since a class becomes more complicated and require more understanding on tester's side.

A higher *Coupling Between Objects* (CBO) count denotes a higher degree of dependencies between classes and affects the reusability and modifiability of the modules.

Ck metrics	**Concept**	**Purpose**
NOC	Services are the key elements	It provides the measures by counting the number of direct descendants
DIT	Reusability, Testability, Understandability	It provides measures for the degree of inheritence.
CBO	Design, Reusability	It determines the degree of dependency among classess and services
LCOM	Design, Reusability	It provides the degree of togetherness of the elements in a class or service
RFC	Design, Testability	It provides measures for the complexity of the class or a service
WMC	Complexity, Reusability	It allows the prediction of maintainability and resuability

Table 2.4: Metrics suite by Keremer [28]

Traditional software metrics like line of code[29] , McCabe's cyclometric complexity[30], Halstead complexity measures[31], Maintainability index[32] and unit test coverage has been widely used to measure the various factors influencing software maintainability.

Metric	**Description**
LOC	Lines of code in a file
Cyclomatic Complexity	Amount of independent execution paths thoughout the code
Halstead Complexity Measures	It comutes the program volume, difficult to understand.
Maintainability Index	It is a single compound value containing Halstead Volume metric, cyclomatic complexity, average lines of code per module and percentage of comment lines per module as elective
Unit test coverage	It shows what portion of the code is covered with automated unit tests

Table 2.5: Traditional metrics [33]

3 Related Work

In this chapter, the existing studies based on coupling, cohesion and maintainability are researched in the form of literature work. Attempting to generalize the software maintainability by comparing two functionally equivalent systems is a challenging task. Therefore no single right solution exists, but various approaches have been proposed. In this chapter first we will introduce the factors influencing maintainability. Then we will go through the different kinds of metrics suggested by the literature to quantify the degree of maintainability, coupling and cohesion. During the literature review, Critiques related to cohesion metric and maintainability are also identified. In the last section of the chapter, existing studies are investigated that compares Service-Based Systemss and Object-Oriented Systems.

In this thesis, the selected metrics are classified into two categories: coupling and cohesion. In the following chapter, maintainability affecting factors are discussed in detail.

3.1 Exploring Maintainability

Reaching the required level of quality is crucial in the software development. Both internal and external quality has played a massive role in the software quality. Software maintainability has been one of the major quality attributes among all the external quality attributes. There is a large body of research that has presented the impact of internal quality attributes on the software maintainability. Internal quality attributes have enabled researchers to predict[34][35] the software maintainability by applying various software metrics to measure the degree of the respective attributes. A large number of metrics have been proposed for the internal quality attributes like size, coupling, cohesion and complexity.

Coupling is defined as the degree of inter dependencies between the software modules while cohesion is the measure of togetherness of the elements that are committed to one and only one task. It has been widely recognized that better quality software should demonstrate higher cohesion and lower degree of coupling[36][37][27].
Internal quality attributes permit software quality to be predicted at an early stage of the software development[36]. Empirical results from various studies have demonstrated that internal attributes coupling and cohesion can have a causal effect on the software maintainability[38][37].

Coupling an cohesion both are contrasting factors since a lower degree of coupling is likely to reduce cohesion in the software modules. Some empirical research evaluated that coupling and cohesion together can influence software maintainability significantly[39].

Analysability, changeability, stability, and testability are the primary sub-characteristics of maintainability that is widely referenced in the literature[40]. Research has stated how cohesion[41] can affect the sub-characteristics of maintainability. Also an Empirical analysis from Mikhail Perepletchikov and Caspar Ryan suggests that highly coupled elements have a negative impact on the analyzability and changeability as compared to loosely coupled elements[34].

3.1.1 Employing software metrics

Structural attributes play a vital role in software's maintainability. They have a profound effect on maintainability. Size, complexity, coupling, and cohesion are widely discussed structural attributes in the literature. Coupling and cohesion attributes are majorly analyzed during the course of this thesis.

Since coupling and cohesion has an impact on software maintainability, therefore, measuring these attribute can help us to evaluate the degree of software maintainability.

The metrics are required to have specific characteristics before they can be successfully applied to the developed systems. These characteristics are comprehensibility, mutual consistency and simplified calculations.

Characteristics	**Purpose**
Comprehensibility	To apply and use accurately, metrics need to be easily interpretable
Simplified calculation	Metrics definitions should be simple in order to perform easy calculations
Mutually consistent	Metrics should be system independent

Table 3.1: Characteristics of metrics

Now we describe the various metrics which were explored from the existing research works but could not be employed due to certain criticism attached with them.
Maintainability index is a metric for predicting software systems maintainability. It is a combination of four distinct metrics which includes Halstead volume metric, the Cyclomatic Complexity, the average count of lines of code per module and optionally the percentage of comment lines per module. The Maintainability index is computed as:

MI = 171 - 3.42ln(aveE) - 0.23aveV(g') - 16.2ln(aveLOC) +(50*sin(sqrt(2.46*aveCM))

It is one of the heavily criticized metrics since MI variations are not clear and there is no substantial evidence regarding the causal relations between the metrics used in the formulation.

Chidamber and Keremer proposed a lack of cohesion in methods(LCOM) metric for the evaluation of cohesion in Object-Oriented Systems[27]. LCOM is powerful at finding non-cohesive classes but its not powerful at classifying between moderate cohesive classes.

Now we define the various coupling and cohesion metrics that are identified for evaluating the degree of coupling and cohesion in service-based and Object-Oriented System.

3.1.1.1 Coupling metrics

Absolute importance of service is determined as the number of other services depending on a given service or calling it's operations.

While Absolute dependence of service is calculated as the dependency of a given service on the number of other services[42].

Degree of coupling is calculated as the proportion of messages received by a class over the count of messages sent by the class.

Degree of coupling(DC) = MRC/MPC

Where MRC is the count of messages received by a class from other classes and MPC is the number of messages a given class passing to other classes[43].

3.1.1.2 Cohesion metrics

Degree of cohesion is calculated as the count of attributes being used in a class to the total attributes for a given class. In Degree of cohesion, the functional strength of the attributes is assessed[44]. It indicates, how powerfully the methods of a class or operations in a service are dependent on the attributes in a class or service.

Degree of cohesion(DCH) = NAU/TNA

where NAU = Number of Attributes Used and TNA = Total Number of Attributes[43].

Tight class cohesion is computed as the ratio of directly connected methods to the maximum number of directly and indirectly connected methods in a given class.

Tight class cohesion(TCC) = NDC/ NP

Where NDC is the no. of directly connected methods and NP is the maximum possible number of methods in a class.

Loose class cohesion is determined by the proportion of directly and indirectly connected methods in a class to the maximum possible number of methods in a class.

Loose class cohesion(LCC) = NDC + NIC/ NP

Where NDC and NIC are the no. of directly and indirectly connected methods and NP is the maximum possible number of methods [45].

3.2 Comparing Service Orientation and Object Orientation Paradigm

Service Orientation is often viewed as an evolution of Object-Orientation as it represents inherited concepts like encapsulation and abstraction. Even if both approaches comply with a different models are adapted to develop semantic models.

Service Orientation is evolved to face the new challenges of growing business requirements nevertheless they balance each other regardless of their dissimilarities Service Orientation still leverages several benefits of Object-Orientation.

This section explores the fundamental similarities and differences between Service Orientation and Object-Orientation followed by further investigation into the existing studies that provide a quantitative comparison based on mature software metrics. These studies mainly assess structural quality attributes and present the effect of Service-Oriented and Object-Oriented paradigms on the coupling, cohesion, size and complexity attributes.

The table 3.2 exhibits the fundamental differences between Service Orientation and Object-Orientation. Basically, objects in Object-Orientation are glued together and work collectively to accomplish business tasks. As a consequence, Object-Orientation is usually practiced for designing inner features of an application whereas Service Orientation promotes the conjunction of several external services where they are loosely coupled.

Contrast	**Object-Orientation**	**Service Orientation**
Concepts	Modelling, Architectural Design, Programming	Modelling, Architectural Design
Focus	Component level	Business level
Communications	Primarily internal	Internal and external
Standards	Extensive standards with high maturity	No standards for specific design patterns
Complexity	Medium to high with a more controlled environment	High, specifically where there is little control over technology

Table 3.2: A contrast between Service Orientation and Object-Orientation paradigms [46]

Service Orientation and Object-Orientation paradigms have different notions about the principles. These paradigms are dependent on different underlying building blocks. Service Orientation relies on services while Object-Oriented is based upon the concepts of objects for the construction of an architectural model.

For a better degree of loose coupling, Service Orientation leans on the service interfaces which separates the services from consumers while Object-Orientation principle of inheritance enforces objects to be tightly coupled for a better degree of reusability.

To impose abstraction in the systems, Service Orientation hides the implementing logic via service interfaces where Object-Orientation aims at separating the object-behavior through the class interface.

Notion	**Service Orientation**	**Object-Orientation**
Underlying block	Services are the key elements	Objects are the key elements
Abstraction	A service hiding the underlying implementation and logic through service interface.	Class hides the state and behaviour of the objects and separates the objects behavior via an interface.
Loose coupling	Service interfaces facilitates loosely coupled environment which separates services from service consumers.	Inheritance enforces objects to be tightly coupled. provides Metrics should be system independent
Statelessness	Services encapsulates business logic and they are stateless since they do not depend on the states of other services.	Objects encapsulates data and behavior. Object's state can be changed by other object's behavior.

Table 3.3: Dissimilarities between Service Orientation and Object-Orientation paradigms [47]

Perepletchikov *et. al.*[48] presented a study that compares Service Orientation and Object-Orientation based on structural quality attributes of size, complexity, coupling and cohesion. They prepared two systems with both the approaches. They designed coarse-grained services based on the principles of Object Orientation (OO) while the Service Oriented System was developed using fine-grained services. Java was used as a primary language for developing both the systems. The results show that Service Orientation presents a lower degree of coupling and permits a better degree of modifiability compared with an Object-Oriented approach. On the contrary, the Object-Oriented approach demonstrates less complexity as compared to a Service-Oriented approach.

Mansour *et. al.* [46] also exhibited a comparative study by assessing the internal software quality attributes of the Service Orientation and Object-Orientation approaches. Size, complexity, cohesion and coupling were the structural attributes under investigation during this case study. The author designed two systems using SO and OO approaches. Both the systems were implemented using C#.NET. The Service-Oriented version was also supported by Web Service Software Factory. The following figure consists of the set of software engineering metrics that have been used for evaluation of the both Service-Oriented and Object-Oriented system.

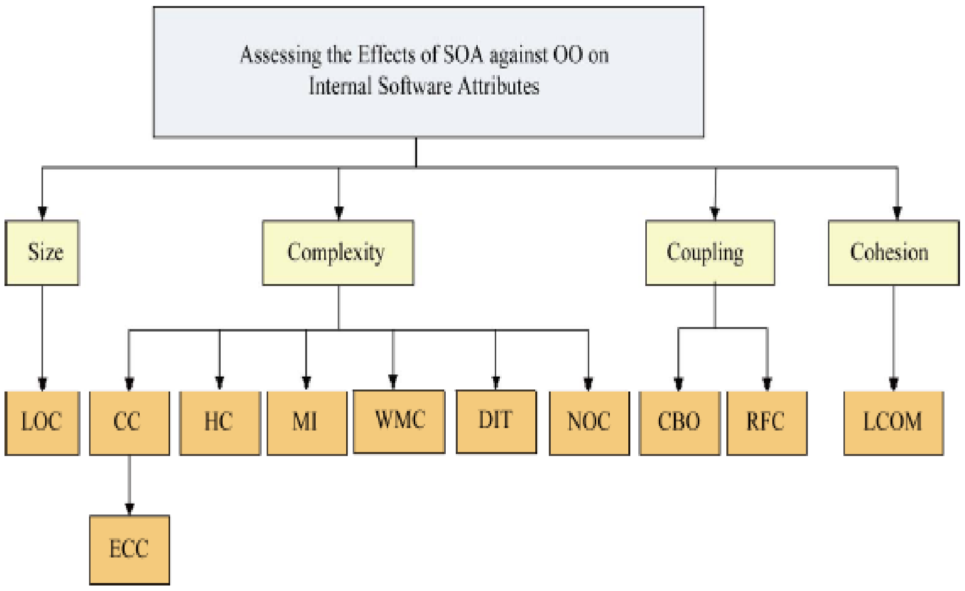

Assessing the Effects of SOA against OO on Internal Software Attributes
Size
Complexity
Coupling
Cohesion
LOC
CC
HC
MI
WMC
DIT
NOC
CBO
RFC
LCOM
ECC

Figure 3.1: Taxonomy of metrics [46]

The case study results show that Service Orientation approach provides higher reusability and lower degree of coupling compared to the Object-Oriented approach. On the other hand, Object-oriented approach indicates lower complexity among modules than Service Orientation approach. Furthermore, this study also reveals that not all the Object-Oriented metrics can be applied to the Service Orientation approach.

Limitations of Existing Case Studies

There are several metrics available for predicting software maintainability but not all Object Oriented or Service Oriented metrics are applicable to the other paradigm. An exploration into a mutually consistent set of metrics can give significant insights into how these paradigms are best applied. Furthermore, there are metrics like Maintainability Index which does not justify the values used for the prediction of software maintainability.

4 Research Approach

This segment prefaces the research approach for the case study. It includes an introduction to the research methodology and the data collection approach.

4.1 Research Methodology

The methodology for this thesis is organized into three phases. The first phase contains the metrics collected by measuring designs/implementation of two functionally equivalent online Book Store systems. Further,

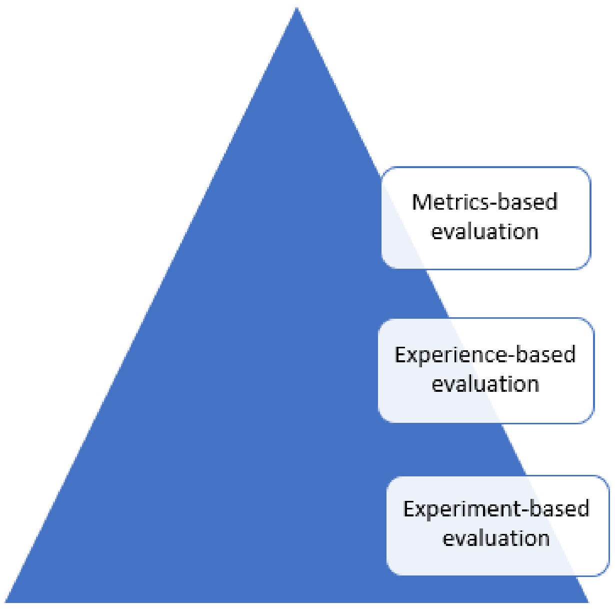


Figure 4.1: Structure of the case study

Experience-based evaluation is the second phase , where expert interviews were conducted in a semi-structured form. It comprised various questions on maintainability of two alternatives, i.e., Service-Oriented and Object-Oriented. Follow-up questions were also included in response to their viewpoints. This phase is also equipped with a survey hosted to gain insights into developers' view on service orientation in comparison to object orientation w.r.t. maintainability.

Third phase is the Experiment-based evaluation where a group of developers were presented with an exercise and requested to perform development on a service-based and Object-Oriented System. Work and efforts of the developers were recorded and investigated for further analysis.

Phase	**Task**	**Reason of selection**
Metrics-based evaluation	Applying software metrics on functionally equivalent SO and OO systems	Comparing degree of maintainability in SO and OO
Experience-based evaluation	Experts interviews(qualitative) survey (quantitative) Survey based on developers view	To gain industry insights into SO and OO
Experiment-based evaluation	A set of developers performing development on two functionally equivalent systems	Work and effort analysis on SO and OO

Table 4.1: Distribution of the phases

4.1.1 Research Strategy

Maintainability has been researched thoroughly but there has been a limited amount of research exists that compares the degree of maintainability into Service Orientation and Object-Orientation.

This case study begins with the factors that impact maintainability of software systems. Later, the software metrics are examined that have been employed to quantify maintainability of service-based and Object-Oriented Systems. Hypotheses and viewpoints would be very significant for further research.

There exist several strategies for this case study, an exploratory form of research strategy is relevant and hence exercised.

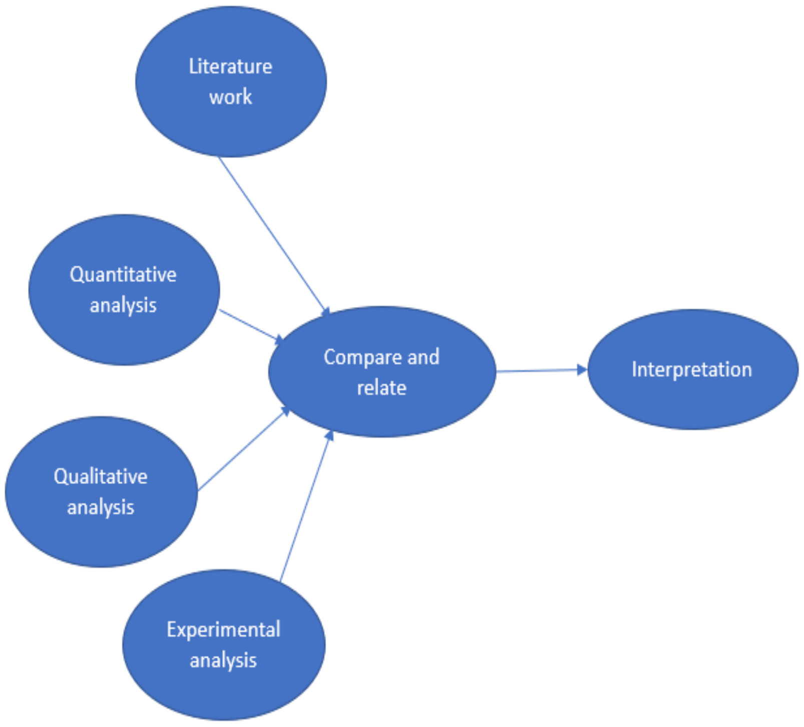


Figure 4.2: An Exploratory strategy

In this process, several research works are explored which provided valuable insights into the topic and helped to generate formal hypotheses for future research work.

4.1.2 Research Design

This case study is of type flexible because not all the parameters were available beforehand. They continued to appear during the course of study.
A fixed design requires a predefined set of elements before data collection can be initiated.

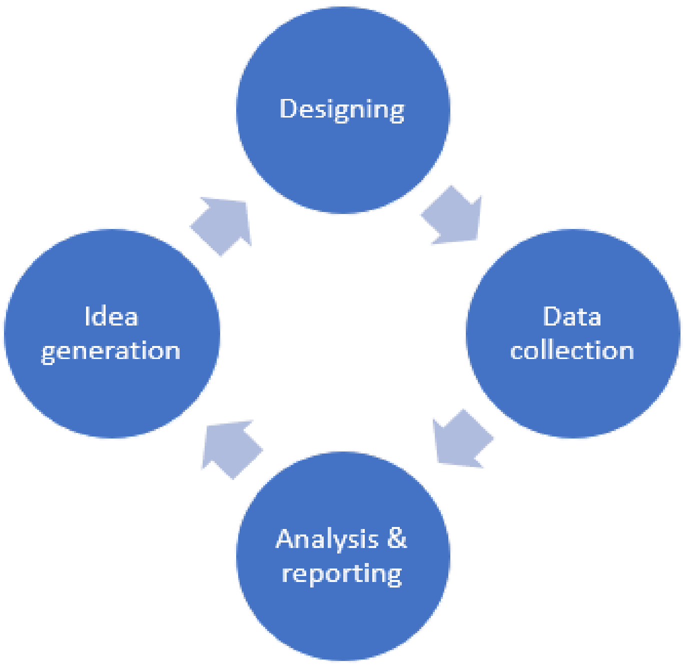


Figure 4.3: Flexible type of research design[49]

4.1.3 Research Category

This case study is a deductive approach where we start with the collection of hypotheses from existing theories and then we test these hypotheses against our observations.

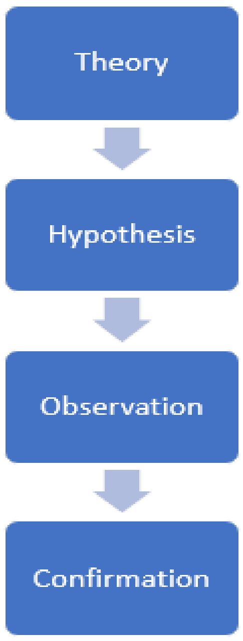


Figure 4.4: Deductive approach[50]

4.2 Research Approach for Data Collection

In this case study, three types of data are collected:
1) Quantitative
2) Qualitative
3) Experimental

Quantitative data is obtained from software metrics measures and the survey questionnaire. These metrics were applied to both service-based and Object-Oriented System. Whereas, the data collected from expert interviews fall under qualitative data.

Expert interviews were conducted in a semi-structured format. These interviews mainly revolve around a set of four questions based on maintainability of two alternatives. It also involved several followup questions.

In the experimental data collection process, A group of developers is invited to perform development on both the systems. Work and efforts on both of the systems were recorded and further analyzed.

Research strategy	Explorative
Research design	Flexible
Research category	Inductive Research
Methods	Metric measures, Experts interviews, Survey, Developmental
Type of data collected	Quantitative, Qualitative, Experimental
Questions types	Open and Close
Type of experts interview	Semi-structured
Arrangement type of experts interview	Personal, Audio and video calling

Table 4.2: Summary of the Research Approach

4.3 Case Study Design

This case study is executed with the mixture of quantitative, qualitative and experimental approaches. Quantitative measures are obtained from coupling and cohesion metrics and a survey questionnaire. Before the implementation of this case study, various literature works were analyzed extensively. The literature study has allowed selecting mutually compatible metrics.

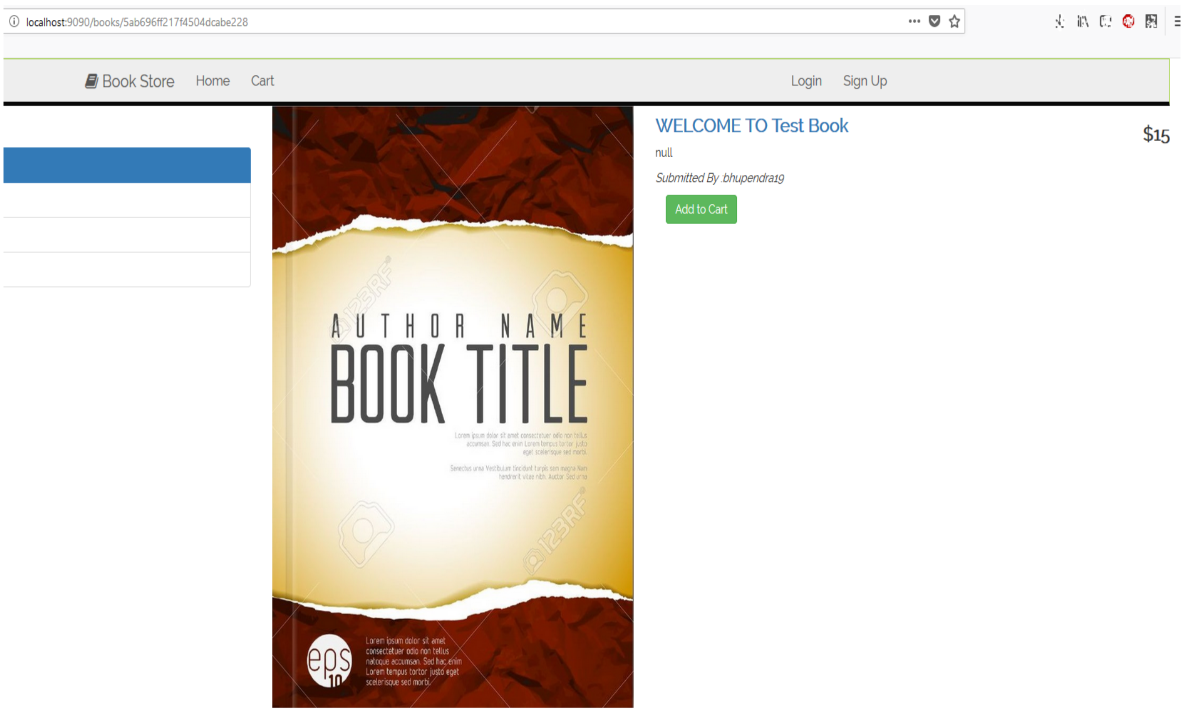
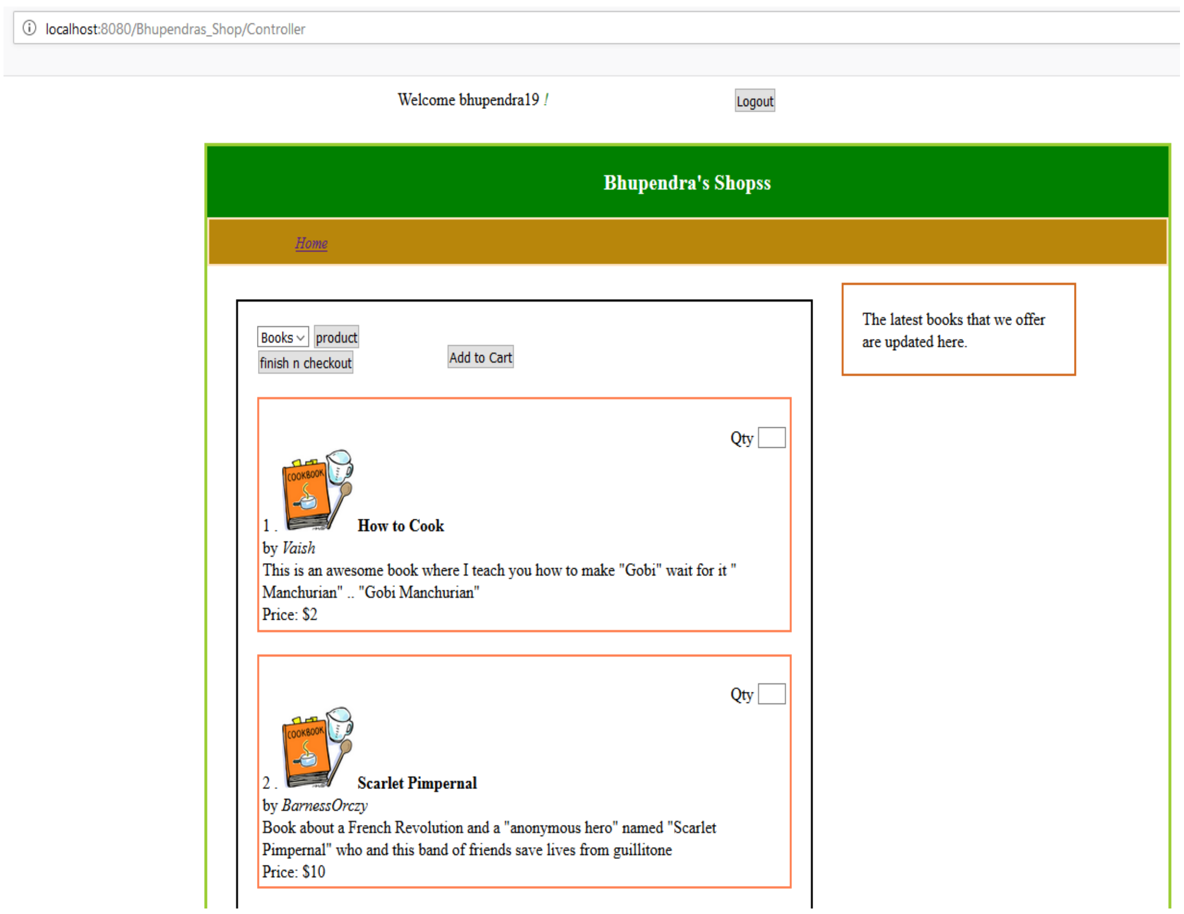


Figure 4.5: A service-based online bookstore

These metrics are basically a collection of coupling and cohesion metrics and applicable to both functionally equivalent systems uniformly. These metrics are applied to both service based and Object-Oriented Systems, and their results are observed in the later section.


localhost:8080/Bhupendras_Shop/Controller

Welcome bhupendra19 !

Bhupendra's Shopss

The latest books that we offer are updated here.

1 . How to Cook
by *Vaish*
This is an awesome book where I teach you how to make "Gobi" wait for it " Manchurian" .. "Gobi Manchurian"
Price: $2

2 . Scarlet Pimpernal
by *BarnessOrczy*
Book about a French Revolution and a "anonymous hero" named "Scarlet Pimpernal" who and this band of friends save lives from guillitone
Price: $10

Figure 4.6: An Object-Oriented online bookstore

An online bookstore(OBS) is designed using a service-based and Object-Oriented approach. It allows registered users to purchase books. These systems have limited functionalities of selecting books and adding them to the shopping carts.

The survey questionnaire is hosted to gain developer's view on Service Orientation in comparison to Object-Orientation w.r.t. maintainability. It is divided into three parts.

First part contains the general questions based on maintainability of software systems. Second part tries to gain the developer's view on the Service-Based Systems and Object-Oriented System.
Last part is aimed at collecting developers opinion on the three different paradigms. It is achieved by obtaining a suitable rank to the paradigms.

Type	**Survey questions**
General analysis of maintainability in software systems	1. Maintainability is an important quality aspect in software systems. 2. Coupling has substantial impact on maintainability. 3. Cohesion has substantial impact on maintainability. 4. Measuring internal software quality brings valuable insights(e.g. coupling, cohesion, size etc).
Comparative analysis of maintainability on Service-based and Object-Oriented Systems	5. In my experience, software based on <paradigm> has a comparatively low degree of coupling. 6. In my opinion, software based on <paradigm> facilitates a comparatively high degree of cohesion. 7. In my experience, software based on <paradigm> promises a significant extent of reusability. 8. In my opinion, software based on <paradigm> possesses a comparatively high level of analyzability. 9. In my experience, software based on <paradigm> reduces the complexity of testing.
Ranking-based analysis of maintainability on software paradigms	10. In your experience, which of the three paradigms provides on average the best degree of modifiability? 11. In your experience, which of the three paradigms provides on average the best degree of encapsulation and abstraction? 12. In your experience, which of the three paradigms provides on average the best manageable size and complexity?

Table 4.3: Survey questionnaire

Qualitative method is implemented in the form of expert interviews.The expert interview mainly revolves around four quality aspects pertained to software maintainability.
These attributes are depicted below:

Quality attributes	Questions
Modifiability(Extensibility and changeability)	In your opinion, What do you think about modifiability in Service-Based Systemss and Object-Oriented Systems? In your experience, which of these systems are best modifiable?
Size and complexity	In your experience, How do you see the size and complexities of Service-Based Systems and Object-Oriented Systems? In your opinion, which of the two alternatives provides best manageable size and complexity?
Analyzability	In your opinion, How do you see the analyzability of two alternatives, i.e. Service-Based Systems and Object-Oriented Systems? Do you think one of them provides a better degree of analyzability?
Stability	In your experience, How do you observe the stability of Service-Based Systems and Object-Oriented Systems? Do you find one of them as a comparatively stable system?

Table 4.4: Scope of experts interviews

Experimental methodology is executed by allowing set of developers performing development work on both service based and Object-Oriented Systems.

They are presented with an exercise of adding new functionality into Service-Based System and object oriented system.

5 Results and Discussion

This section presents the results and findings of the quantitative, qualitative and experimental approach mentioned in the above case study section.

5.1 Quantitative Results

Quantitative results are collected in three phases as stated in the figure . In the first phase, metrics results and data from survey questionnaire are gathered. These metrics results and survey data are transformed into graphs for comparative analysis.

Last phase represents the findings obtained from various charts and results.

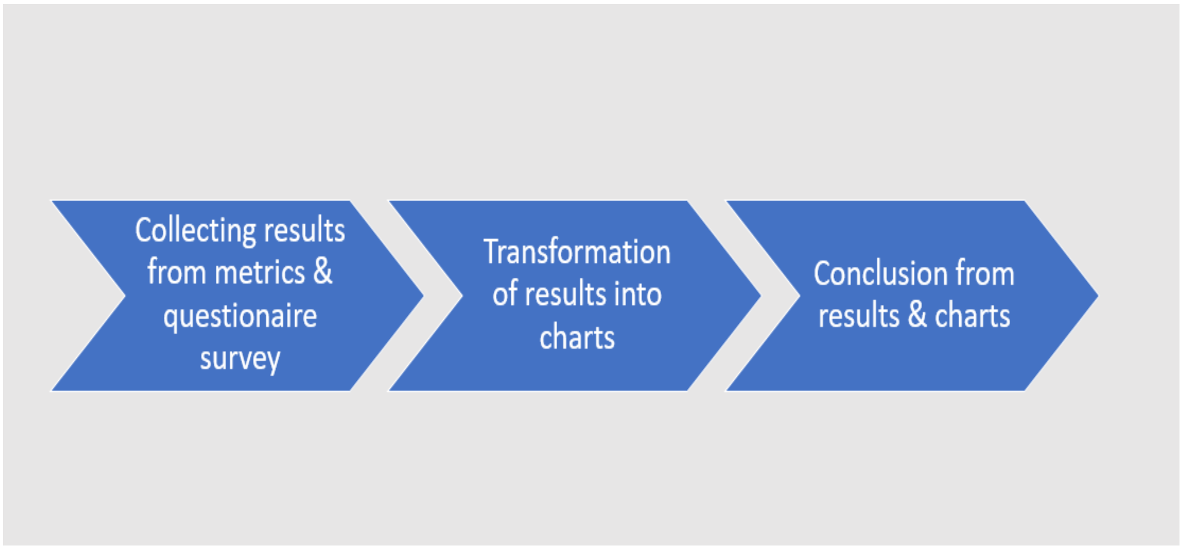

Collecting results from metrics & questionaire survey
Transformation of results into charts
Conclusion from results & charts

Figure 5.1: Phases in the collection of quantitative results and findings

5.1.1 Metrics Collection

Several metrics such as maintainability, coupling, cohesion are analyzed through the literature study. As a result, a set of coupling and cohesion metrics are selected. They are chosen by their mutual consistency towards service based and Object-Oriented System.

These metrics are applied to the functionally equivalent designed systems.

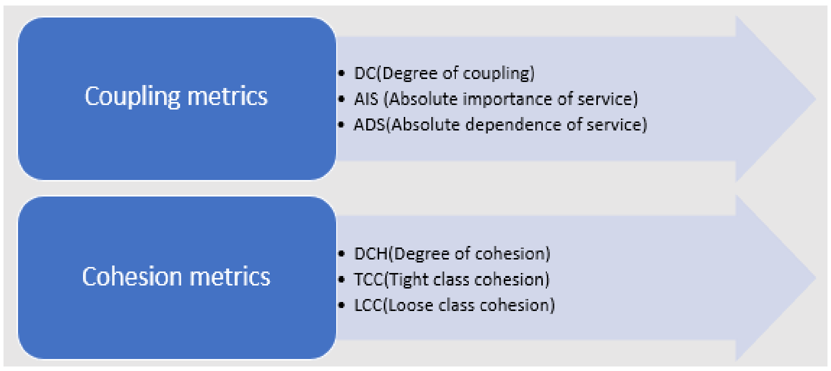

Coupling metrics
DC(Degree of coupling)
AIS (Absolute importance of service)
ADS(Absolute dependence of service)
Cohesion metrics
DCH(Degree of cohesion)
TCC(Tight class cohesion)
LCC(Loose class cohesion)

Figure 5.2: coupling and cohesion metrics

The metric measures are collected from the designed online bookstore(OBS) system displayed in the figure 4.5 and 4.6.
Since the software metrics are primarily designed for Object-Oriented Systems, they are not all exercisable for Service-Based Systems. So, following supposition is made to measure services using software metrics.

Service = Class

It is worth taking a note that neither node libraries from Service Oriented version nor java libraries from Object Oriented were taken into account during measurements. Only direct implementation artifacts were quantified.

5.1.1.1 Coupling in Service-Based System

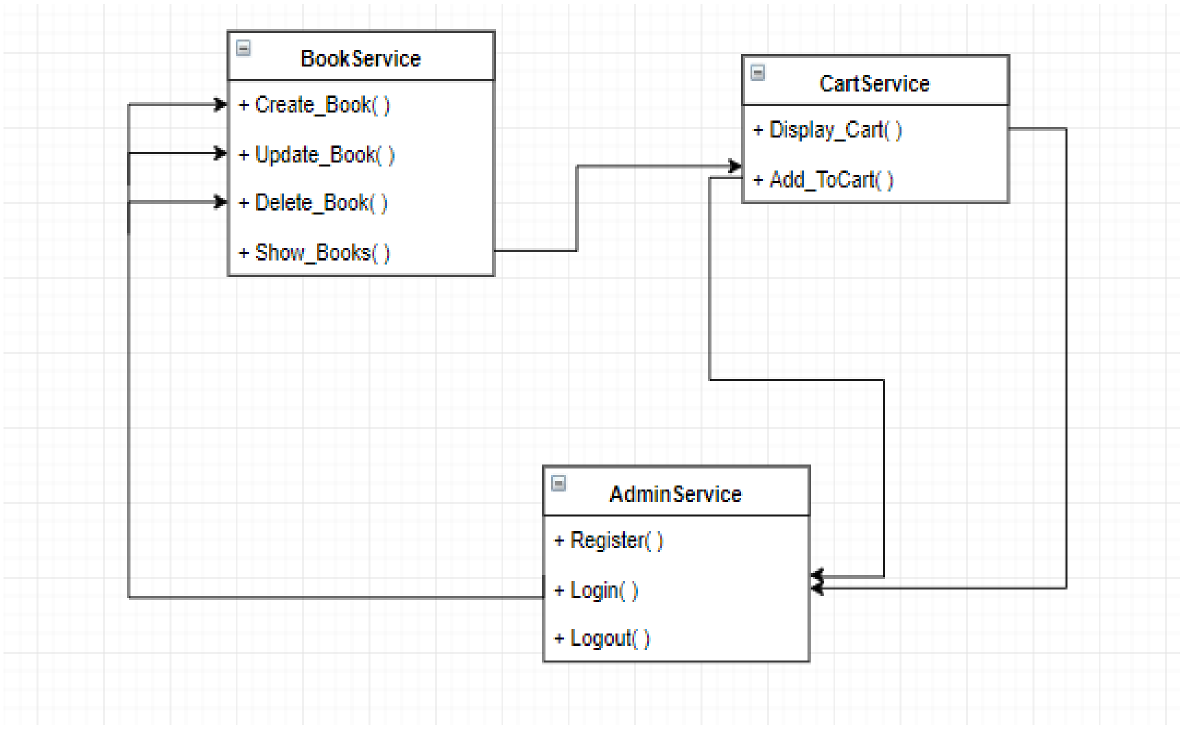


Figure 5.3: Coupling in a Service-Based System

The degree of coupling [51] for a service based system is calculated as follows:
BookService has four operations. It is sending a message to CartService and receiving messages from AdminService. Therefore, message passing service (MPS) for this service is 1 and message receiving service(MRS) is 3. Hence, the degree of coupling can be calculated as:

Degree of coupling(DC) = MRS/MPS

DC(BookService)= 3/1 = 3

CartService has two operations and it has sent messages to AdminService therefore MPS for this service is 2. Further, this service is receiving a message from BookService, so MRS for this service is 1. Therefore the degree of coupling can be given as:

DC(CartService) = 1/2 = 0.5

AdminService contains three operations. It is passing messages to the BookService, so MPS for AdminService is 3. Furthermore, it is receiving messages from CartService, so MRS is 2. Therefore the degree of coupling can be determined as:

DC(AdminService) = 2/3 = 0.66

Absolute dependence of service(ADS) is calculated as the count of other services, a given service depend on.
Absolute dependence of service(ADS) in a Service-Based System is computed as:

BookService(ADS) = 1
CartService(ADS) = 1
AdminService(ADS) = 1

Absolute importance of service(AIS) is determined as the count of other services which depends on a giver service or which invokes its operations.

Absolute importance of service(AIS) in a Service-Based System can be calculated as:

BookService(AIS) = 1
CartService(AIS) = 1
AdminService(AIS) = 1

The following table provides the summary of coupling metrics for a Service-Based Systems:

Service	**DC**	**AIS**	**ADS**
BookService	3	1	1
CartService	0.5	1	1
AdminService	0.66	1	1

Table 5.1: Summary of coupling metrics for Service-Based System

5.1.1.2 Cohesion in Service-Based System

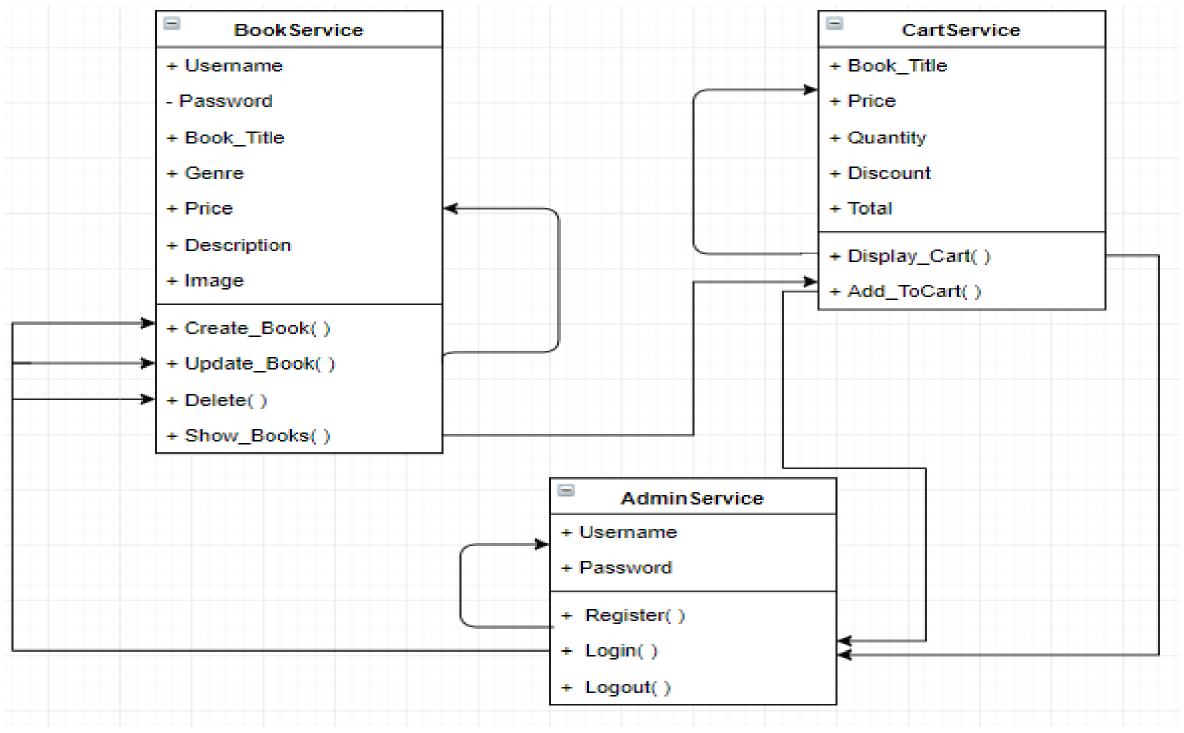


Figure 5.4: Attributes Calling Graph for a Service-Based System

The degree of cohesion metric takes in account of the functional strength of the associated attributes within a given class or service.

Here the Number of attributes used are the ones that contribute to the functional behavior of a service while the total no. of attributes shows the available attributes for a given service or class.
For calculating DCH, attribute calling graph(ACG) is designed as demonstrated in the figure. This graph shows the connectivity between the services and the number of attributes that are used by the operations of a service.

While in case of an Object Oriented system the same applies on a class level where we have classes and methods in place of services and operations.

Hence, The degree of cohesion(DCH) of a Service-Based System is calculated in the following table:

Service	**No. of attributes used**	**Total no. of attributes**	**Degree of cohesion**
Book	5	7	5/7
Cart	5	5	1
Admin	2	2	1

Table 5.2: Degree of cohesion in Service-Oriented System

Tight service cohesion for Service-Based System is computed as follows:
BookService(TSC) = 2/6 = 0.33
CartService(TSC) = 2/2 = 1
AdminService(TSC) = 2/3 = 0.66

Loose service cohesion
BookService(LSC) = 3/6 = 0.50
CartService(LSC) = 2/2 = 1
AdminService(LSC) = 2/3 = 0.66

5.1.1.3 Coupling in a Object-Oriented System

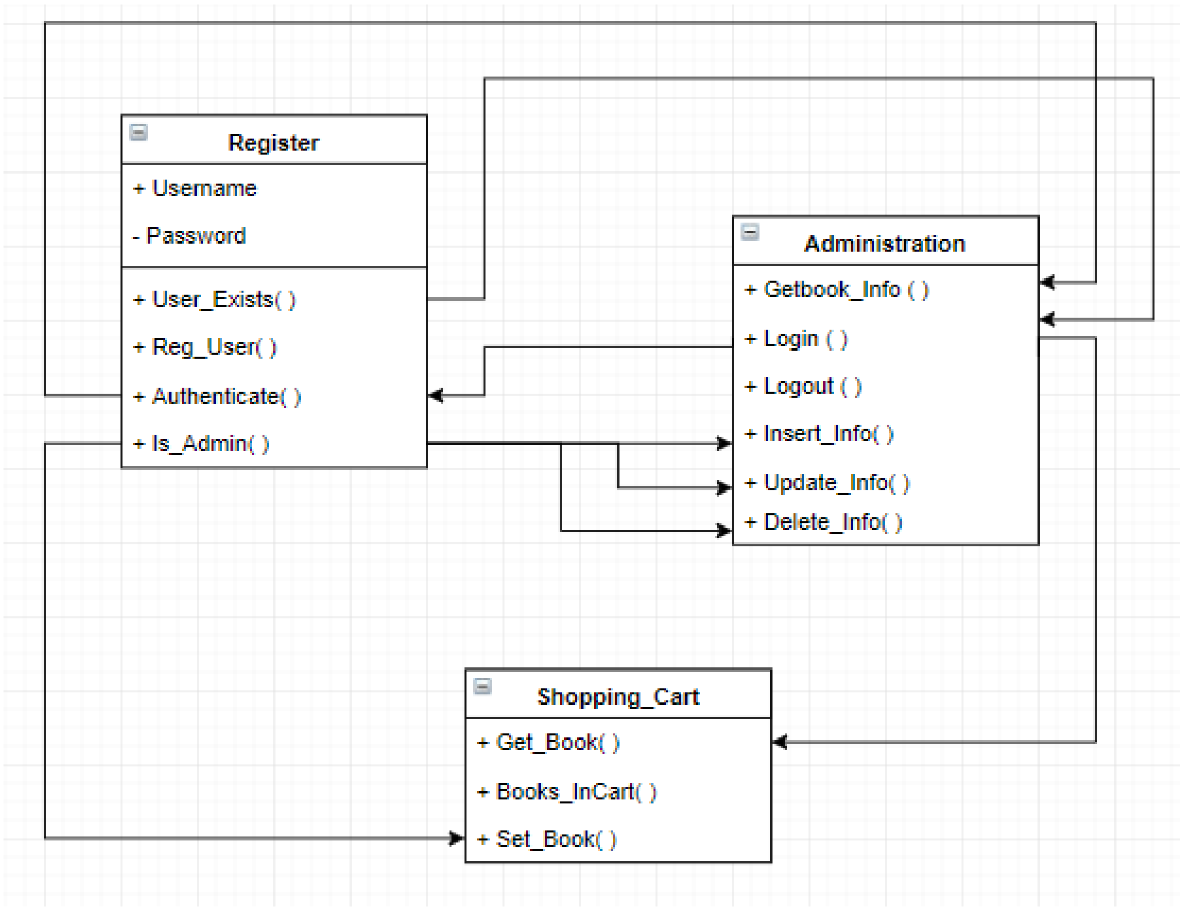


Figure 5.5: Coupling in a Object-Oriented System

Degree of coupling for a Object-Oriented System can be calculated as following: RegisterClass contains four methods. It sends messages to the both of the classes, administration and shoppingcart.Furthermore, It receives messages by AdministrationClass only. Therefore, message passing class (MPC) for this class is 6 and message receiving class(MRC) is 1. Hence, the degree of coupling can be calculated as:

Degree of coupling(DC) = MRC/MPC

DC(RegisterClass)= 1/6 = 0.16

Administration class has six methods. It is passing messages to both Register and ShoppingCart Classes but receiving messages from RegisterClass only. Therefore, MPC for this class is 2 and MRC is 5. Thus the degree of coupling can be given as:

DC(AdminClass) = 5/2 = 2.5

ShoppingCart class consists of three methods and it receives messages from both the other classes but doesn't send any messages. Hence MPC for this class is 0 and MRC is 2.
Thus the degree of coupling can be calculated as:

$$DC(ShoppingCart) = 2/0 = \infty$$

Absolute dependence of class(ADC) in a Object-Oriented System is computed as:
RegisterClass(ADC) = 2
AdministrationClass(ADC) = 2
ShoppingCartClass(ADC) = 0

Absolute importance of class(AIC) in a Object-Oriented System can be calculated as:
RegisterClass(AIC) = 1
AdministrationClass(AIC) = 1
ShoppingCartClass(AIC) = 2

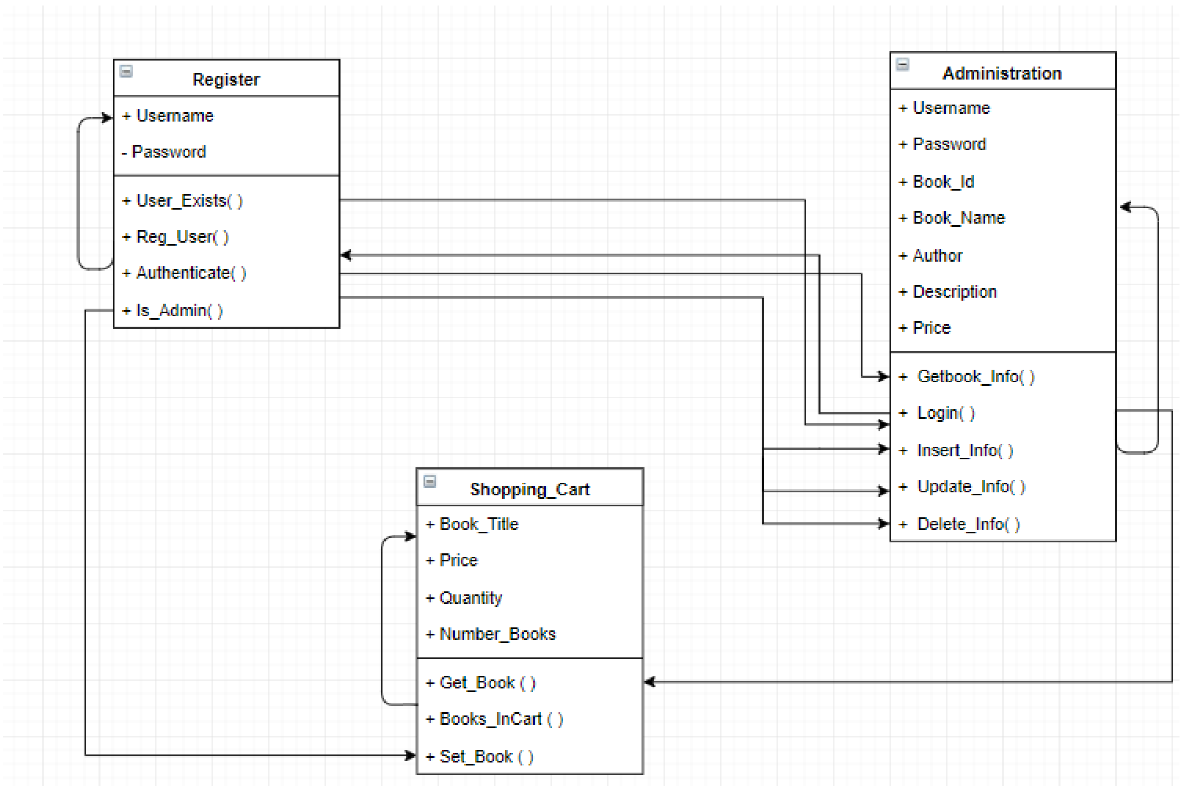


Figure 5.6: Attribute Calling Graph for a Object-Oriented System

The degree of cohesion of a Object-Oriented System is calculated in the following table:

Class	**No. of attributes used**	**Total no. of attributes**	**Degree of cohesion**
Register	2	2	1
Cart	2	4	1/2
Admin	5	7	5/7

Table 5.3: Degree of cohesion in Object-Oriented System

Tight class cohesion in Object-Oriented is computed as:
RegisterClass(TCC) = 1/6 = 0.16
AdministrationClass(TCC) = 0/15 = 0
ShoppingCartClass(TCC) = 1/3 = 0.33

Loose class cohesion in Object-Oriented is calculated as:
RegisterClass(LCC) = 3/6 = 0.50
AdministrationClass(LCC) = 6/15 = 0.40
ShoppingCartClass(LCC) = 1/3 = 0.33

5.1.1.4 Discussion

Before applying software metrics to the service based and Object-Oriented Systems, an informal hypothesis and analysis were drawn from literature research. This hypothesis and the analysis were assessed against the measures collected from a set of coupling and cohesion metrics.

Metric measures collected from OBS are presented in the table 5.4. These measures were obtained by applying a set of coupling and cohesion metrics on the designed service-based and Object-Oriented Online BookStore. Values for The degree of coupling(DC) metrics are not used since they are not suitable for comparison purpose.

System type	AIS_{AVG}	ADS_{AVG}	DCH_{AVG}	TCC_{AVG}	LCC_{AVG}
OO	1.33	1.33	0.73	0.16	0.41
SO	1	1	0.90	0.76	0.72

Table 5.4: Metrics values collected by measuring Service-Oriented and Object-Oriented OBS

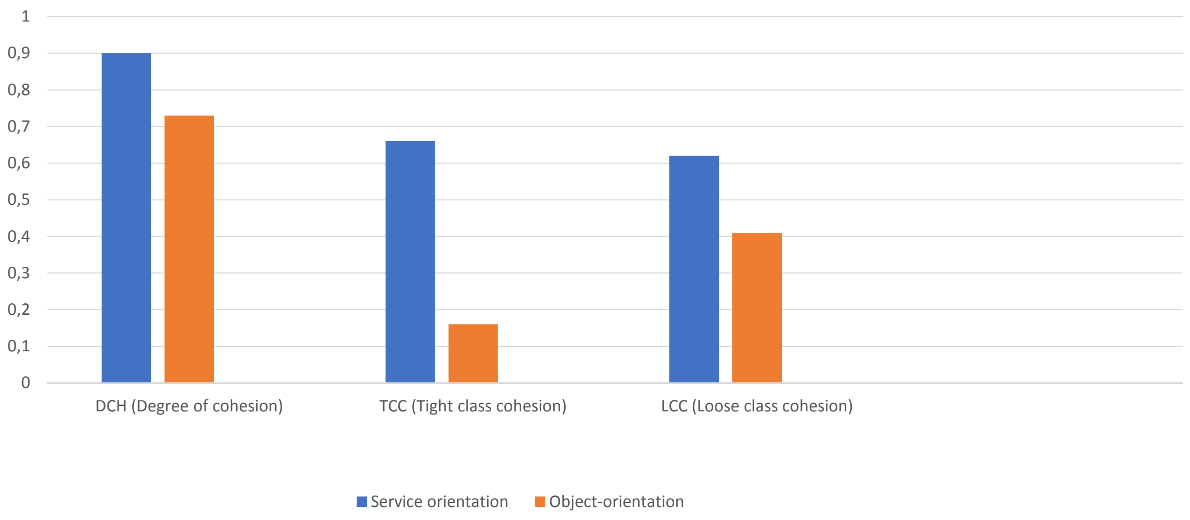


Figure 5.7: Comparing cohesion between Service-Based and Object-Oriented Online Book-Store(OBS)

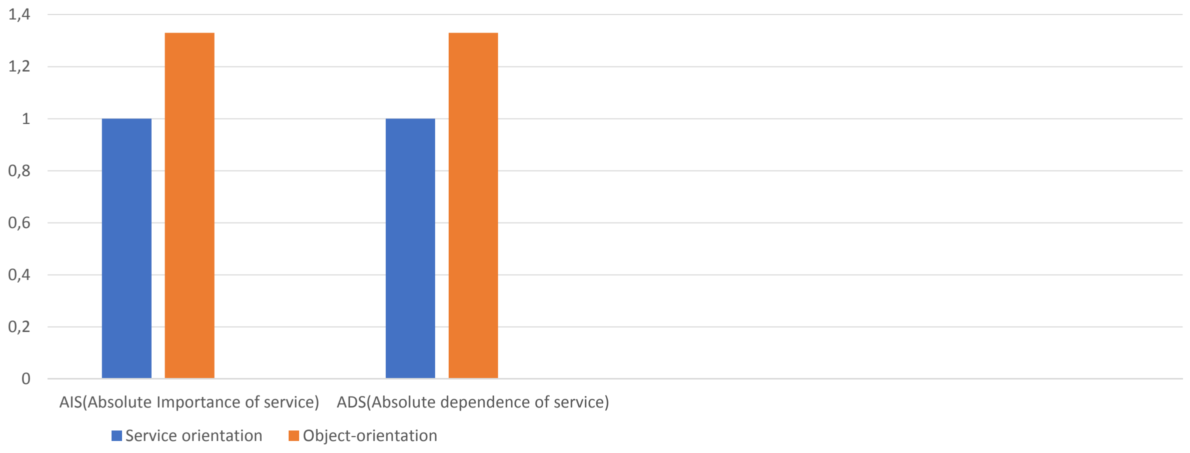


Figure 5.8: Comparing coupling between Service-based and Object-Oriented Online Book-Store(OBS)

H1: Systems developed using Service Orientation approach indicates lower degree of coupling as compared to the systems developed using Object-Orientation approach since there is no strong connection between the services in a system developed using Service Oriented approach [46, 48].

Mansour and Mustafa [46] analyzed that Systems developed using Service Orientation approach exhibit higher degree of cohesion compared to the systems developed using Object-Oriented approach since the business logic is encapsulated in the business constituents itself.

Some general observations can be formed regarding the metrics results concerning the above mentioned informal hypothesis and analysis.

Firstly, The AIS and ADS metrics show a bit high values for OO than a Service-Based System. The Service-Based System presents a bit lower degree of coupling compared to an Object-Oriented System and hence favoring hypotheses 1.

Secondly, The DCH, TCC and LCC counts are higher for a Service-Based System compared to OO approach. Hence an observation can be made that Service-Based System indicates a higher degree of cohesion compared to the traditional methods like OO as anticipated by the analysis.

5.1.2 Questionnaire Survey

Online survey tool SmartSurvey [52] provided a platform to host survey questionnaire. The survey consists of twelve questions divided into three parts. Survey link is shared among the people having relevant development experience with service-based and Object-Oriented Systems. It includes personal contacts and was openly distributed within industry.
The survey is summarized in the table below:

Total no. of questions	12
Start date of the survey	19.04.2018
End date of the survey	06.05.2018
Total participants	33
Participants type	Developers with relevant expertise

Table 5.5: Summary of the survey questionnaire

Total 32 participants were able to finish the survey questionnaire. Based on various aspects of software maintainability, Three different sets of questions were prepared. It enabled us to interpret the notion of maintainability in service-based and Object-Oriented Systems.

Survey results are divided into three parts. The first part consists of four questions. Their results are displayed one by one below.

1. Importance of maintainability

As can be seen from figure 5.9 , over 90% of the respondents reported software maintainability as an important quality aspect.

2. Impact of coupling on maintainability

As per results from figure 5.10, 29 out of 32 respondents believe that coupling in software system impacts software maintainability substantially.

3. Impact of cohesion on maintainability

Around half of the respondents reported cohesion as an influencing factor on software maintainability. This can be seen in figure 5.11.

4. Importance of software quality measurements

Majority of the respondents(29) considered internal quality measures as an important aspect of software systems. Figure 5.12 depicts the same results.

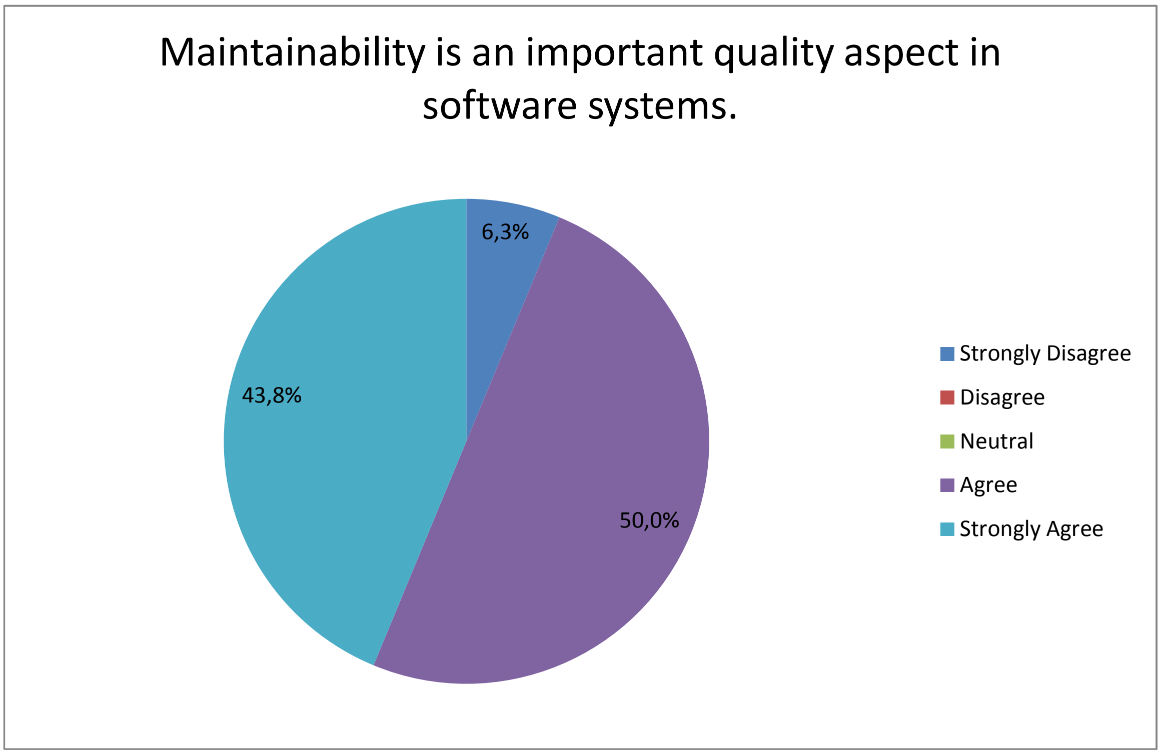


Figure 5.9: Importance of software maintainability

Answer Choice		Response Percent	Response Total
1	Strongly Disagree	6,3%	2
2	Disagree	0,0%	0
3	Neutral	0,0%	0
4	Agree	50,0%	16
5	Strongly Agree	43,8%	14
		answered	**32**
		skipped	**0**

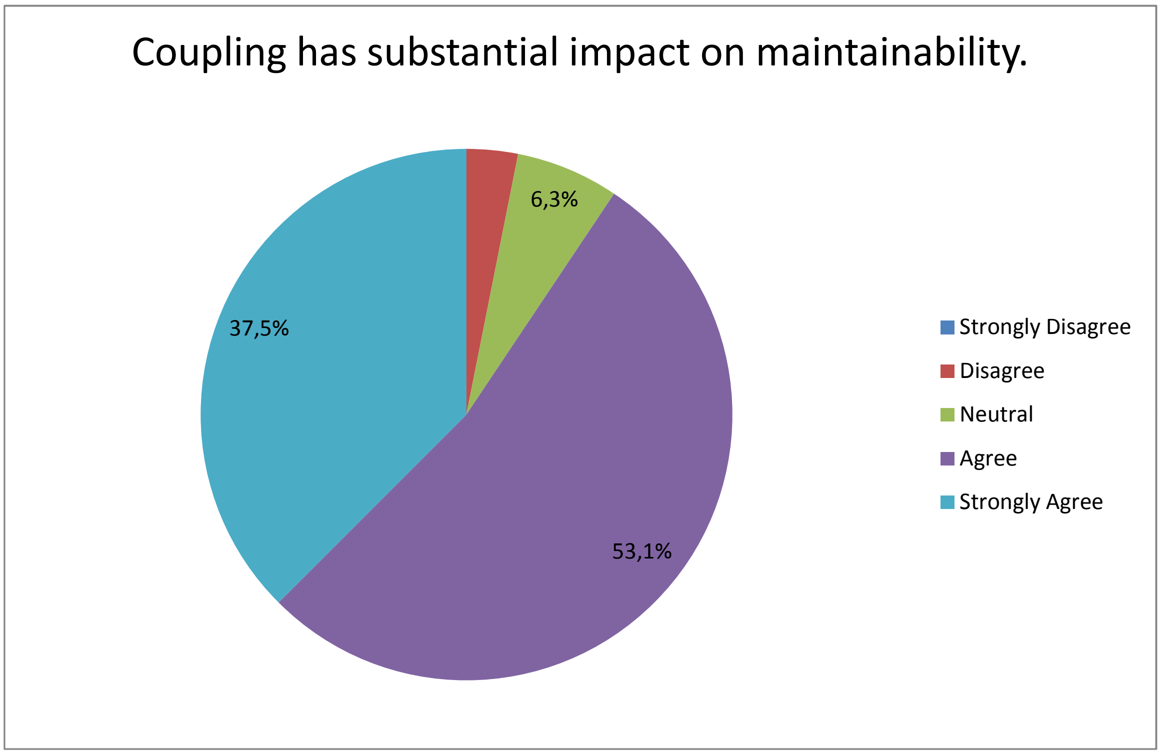


Figure 5.10: Impact of coupling on maintainability

Answer Choice		Response Percent	Response Total
1	Strongly Disagree	0,0%	0
2	Disagree	3,1%	1
3	Neutral	6,3%	2
4	Agree	53,1%	17
5	Strongly Agree	37,5%	12
		answered	**32**
		skipped	**0**

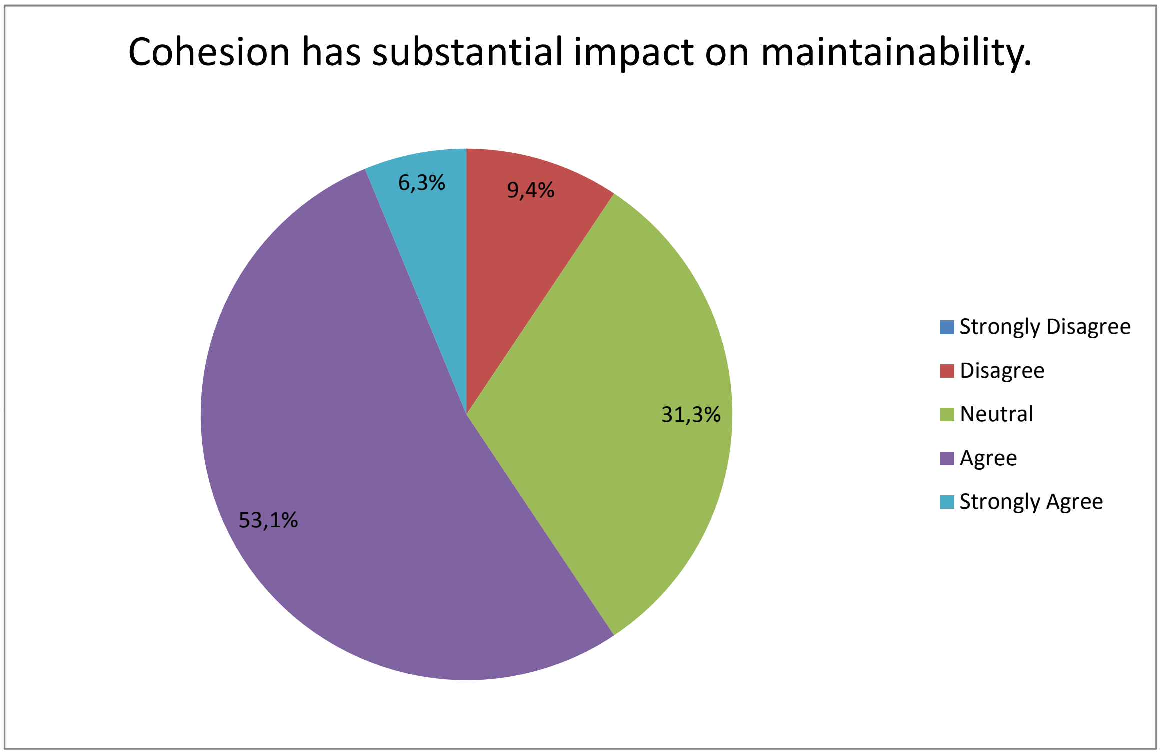


Figure 5.11: Impact of cohesion on maintainability

Answer Choice		Response Percent	Response Total
1	Strongly Disagree	0,0%	0
2	Disagree	9,4%	3
3	Neutral	31,3%	10
4	Agree	53,1%	17
5	Strongly Agree	6,3%	2
		answered	**32**
		skipped	**0**

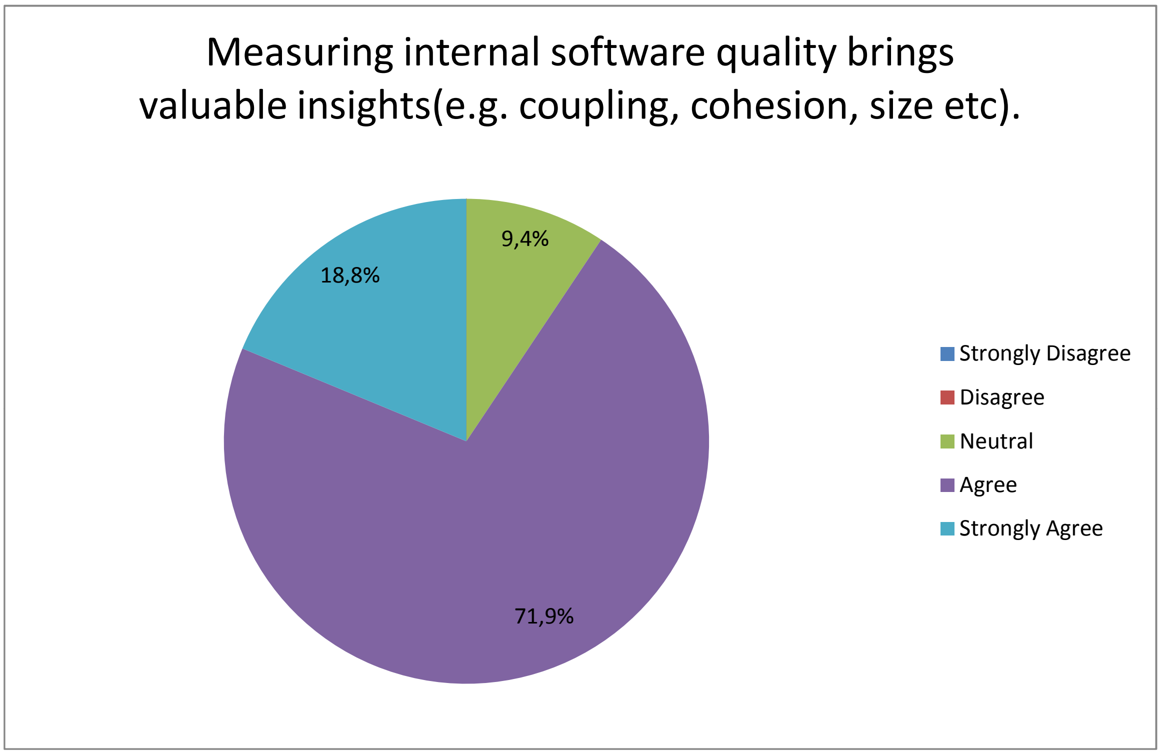


Figure 5.12: Importance of software quality measurements

Answer Choice		Response Percent	Response Total
1	Strongly Disagree	0,0%	0
2	Disagree	0,0%	0
3	Neutral	9,4%	3
4	Agree	71,9%	23
5	Strongly Agree	18,8%	6
		answered	**32**
		skipped	**0**

The second part of the survey results provides the notion of maintainability of a Service-Based System and an Object-Oriented System. It consists of five questions, which are presented below:

5. Notion of coupling in SBS and OO

As visible from the figure **??**, Over 90% of the respondents observed Service-Based Systems as comparatively loosely coupled systems while only half of the respondents considered Object-Oriented as loosely coupled systems compared to SBS.

6. Notion of cohesion in SBS and OO

Around 63% of the participants informed Service-Based Systems as comparatively highly cohesive systems while up to 50% of the participants viewed Object-Oriented System facilitating a higher degree of cohesion.

7. Reusability in SBS and OO

25 participants identified that a Service-Based System promises comparatively a better degree of reusability where in case of Object-Orientation 19 participants reported better reusability.

8. Analyzability level in SBS and OO

In this maintainability aspect, Respondents reported Service-Based Systems as better analyzable compared with Object-Oriented Systems..

9. Testing complexity in SBS and OO

Above 70% of the respondents observed that Object-Oriented Systems possess a comparatively reduced testing complexity.

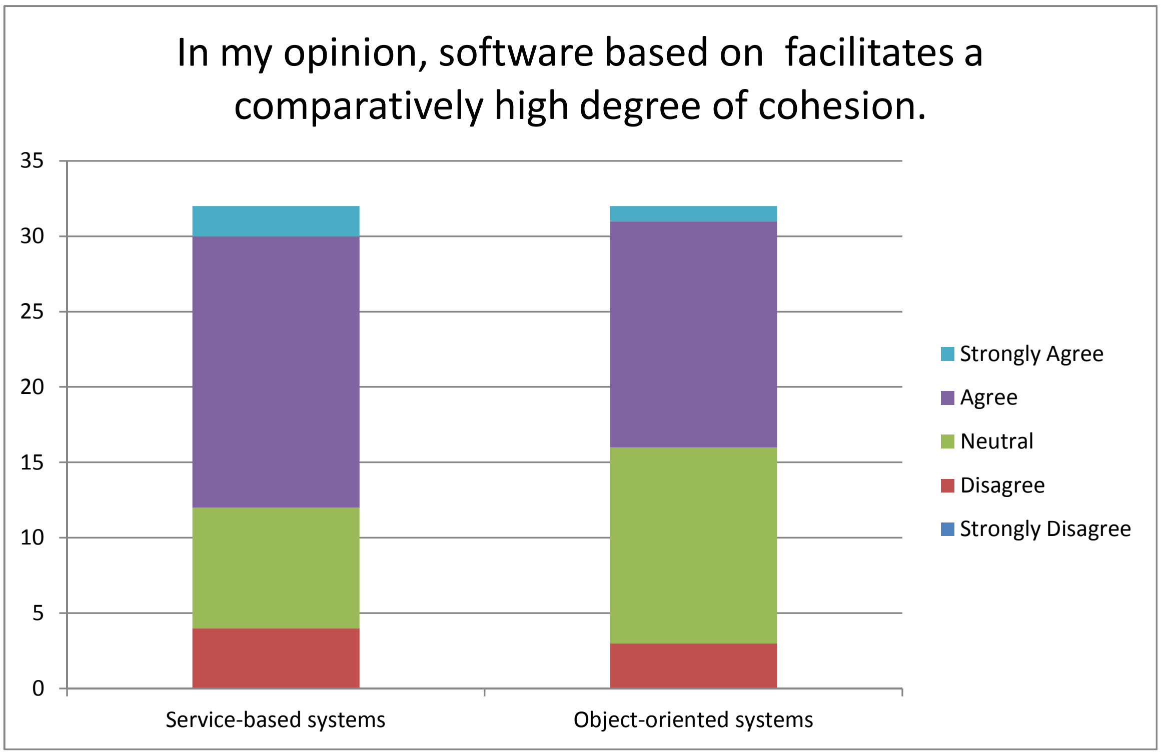


Figure 5.13: Notion of cohesion in SBS and OO

Answer Choice		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Response Total
1	Service-based systems	0	4	8	18	2	32
2	Object-oriented systems	0	3	13	15	1	32
						answered	**32**
						skipped	**0**

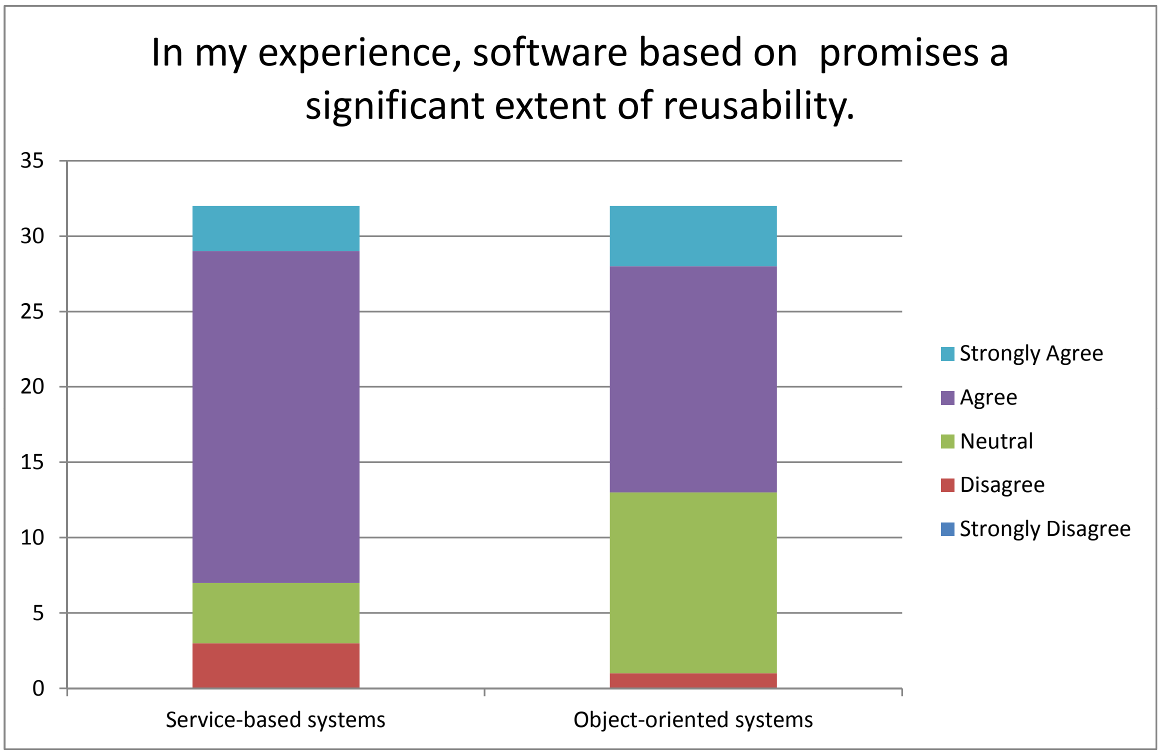


Figure 5.14: Reusability in SBS and OO

Answer Choice		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Response Total
1	Service-based systems	0	3	4	22	3	32
2	Object-oriented systems	0	1	12	15	4	32
						answered	**32**
						skipped	**0**

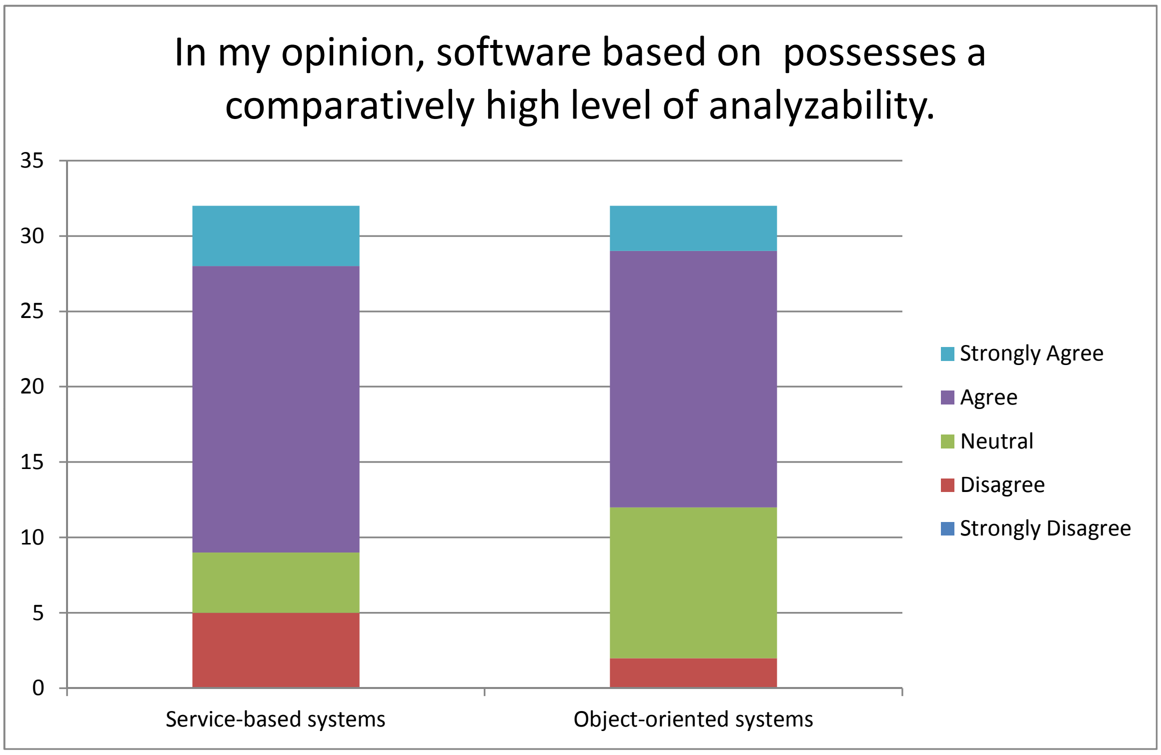


Figure 5.15: Analyzability level in SBS and OO

Answer Choice		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Response Total
1	Service-based systems	0	5	4	19	4	32
2	Object-oriented systems	0	2	10	17	3	32
						answered	**32**
						skipped	**0**

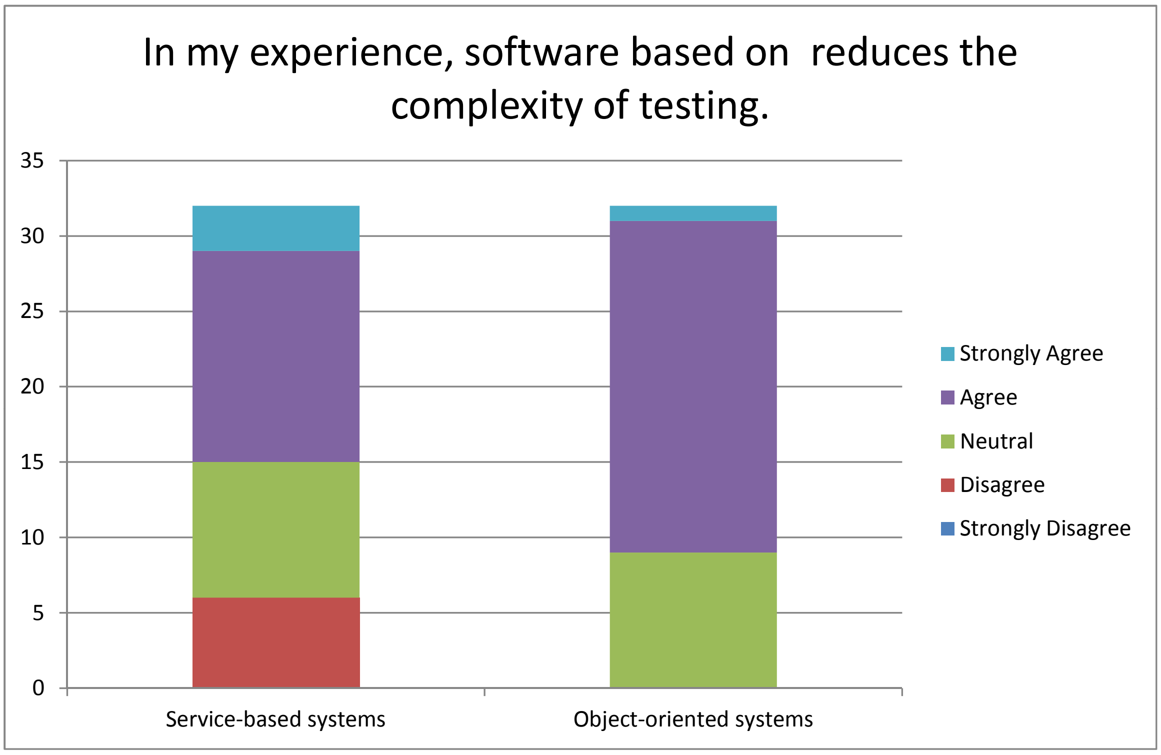
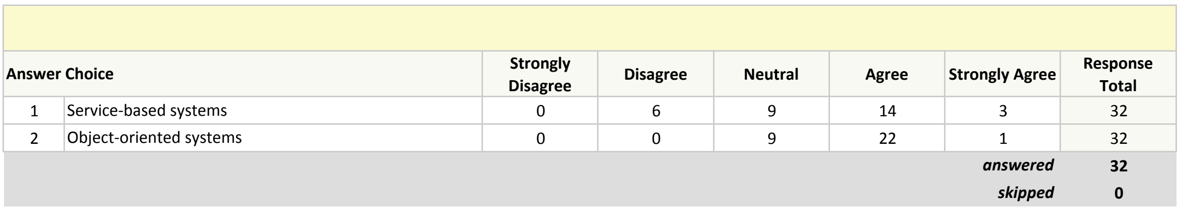


Figure 5.16: Testing complexity in SBS and OO



Answer Choice		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Response Total
1	Service-based systems	0	6	9	14	3	32
2	Object-oriented systems	0	0	9	22	1	32
						answered	32
						skipped	0

The last part provides the ranking based analysis for the Service-Based, Object-Oriented, Component-Based paradigms. It contains three questions and their results are outlined below:

10. Modifiability

Majority of the respondents ranked Service-Based Systems as better modifiable systems while Object-Oriented Systems secured the second position in this ranking based analysis. The figure 5.17 shows the results collected from the analysis.

11. Encapsulation and abstraction

Most of the participants identified service based system with a better degree of encapsulation and abstraction where Object-Orientation appeared to be the second best option in the process. The same can be observed from the figure 5.18.

12. Size and complexity

As noticeable from the figure 5.19, Participants observed that Object-Oriented Paradigm provides best manageable size and complexity where Service Orientation appeared to be the next best option in the process. However their differences were observed in a small extent.

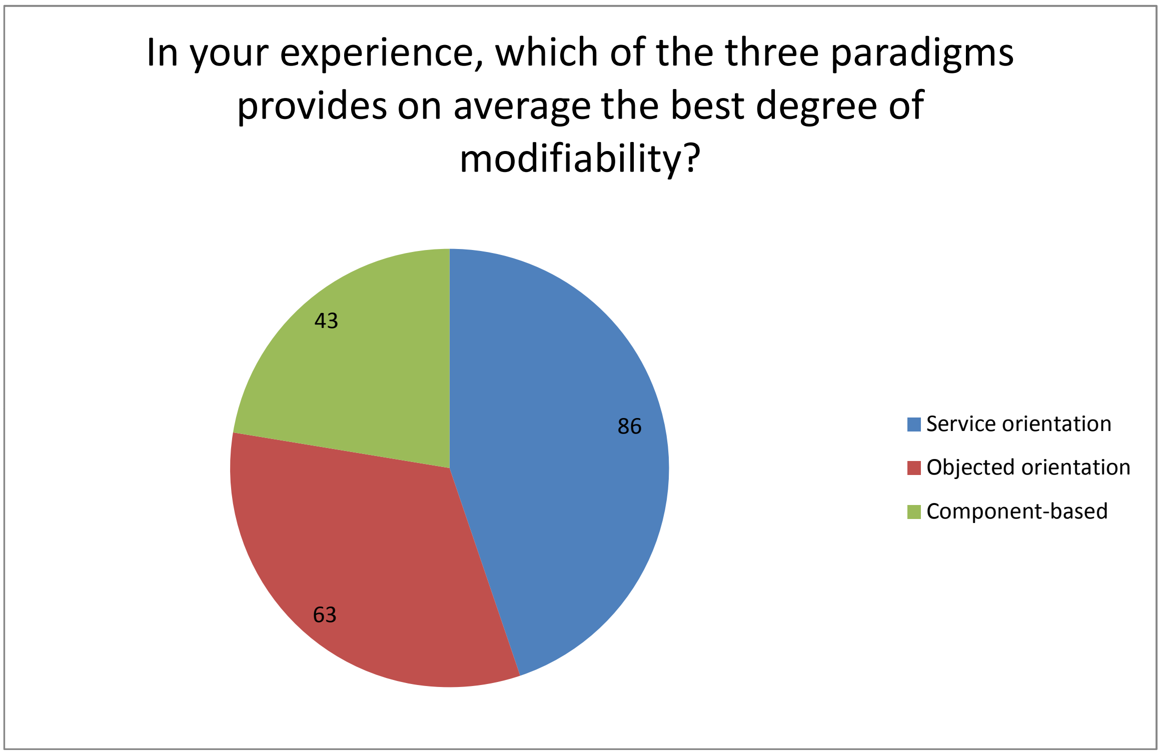


Figure 5.17: Notion of modifiability in the paradigms

Answer Choice		Total Score	Overall Rank
1	Service orientation	86	1
2	Objected orientation	63	2
3	Component-based	43	3
		answered	**32**
		skipped	**0**

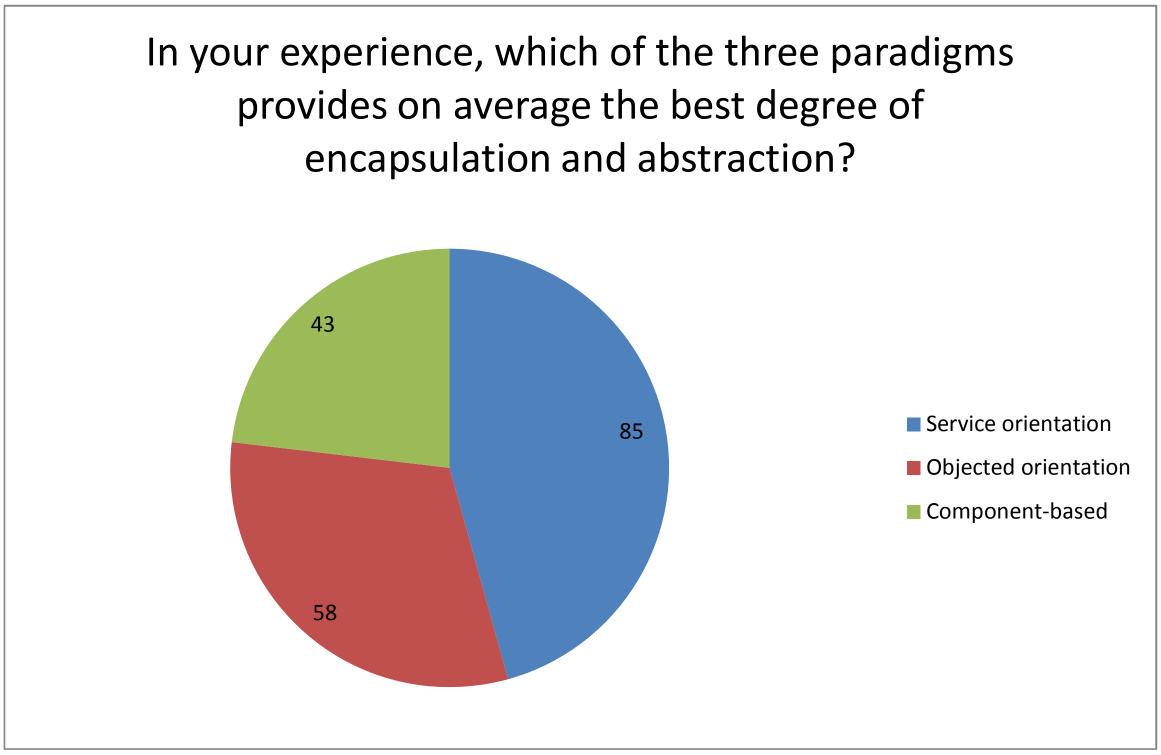


Figure 5.18: Encapsulation and abstraction of the paradigms

Answer Choice		**Total Score**	**Overall Rank**
1	Service orientation	85	1
2	Objected orientation	58	2
3	Component-based	43	3
		answered	**31**
		skipped	**1**

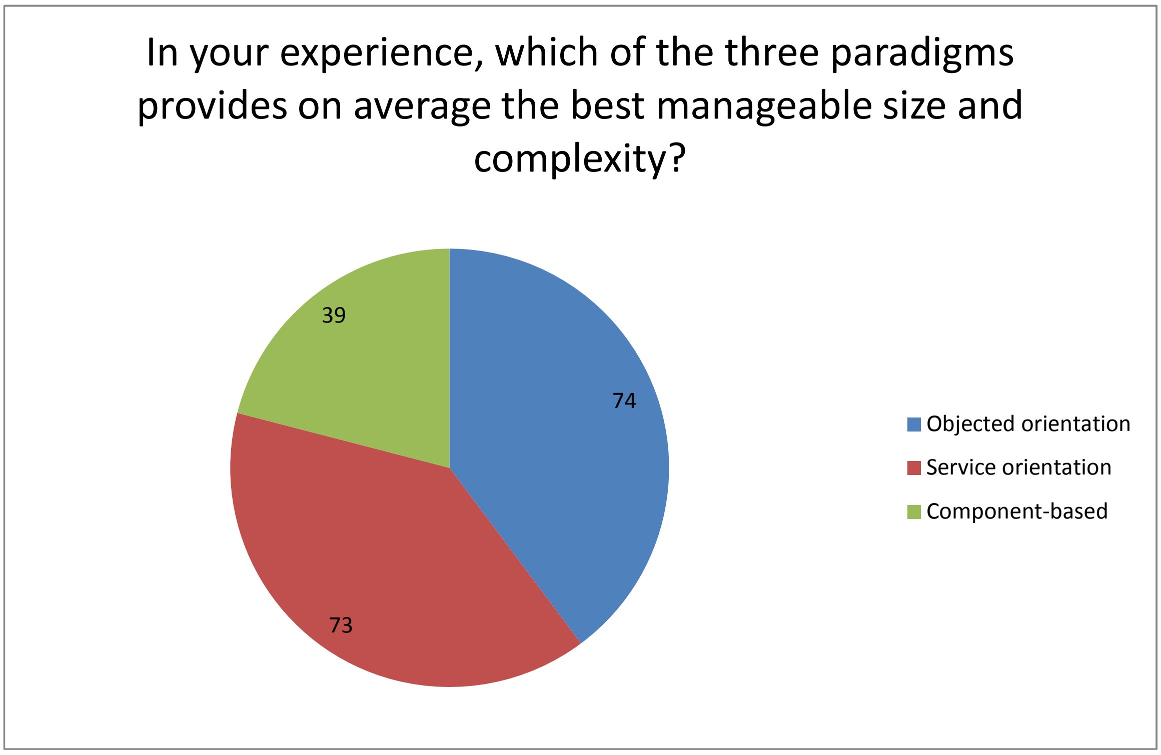


Figure 5.19: Size and complexity of the paradigms

Answer Choice		**Total Score**	**Overall Rank**
1	Objected orientation	74	1
2	Service orientation	73	2
3	Component-based	39	3
		answered	**31**
		skipped	**1**

Discussion

In the survey, above 90% of the respondents feel that maintainability is an essential aspect for keeping software quality intact. They believe that coupling has a substantial impact on the software maintainability while cohesion has a comparatively lower impact.

From the figure 5.12, participants believe that measuring internal quality such as size, complexity and coupling can provide valuable insights on software maintainability.

Comparative analysis of maintainability between SBS and OO
Developers assume that Service-Based Systems promise a relatively lower degree of coupling and a higher degree of cohesion. Moreover, It is observed that SBS indicates a higher degree of reusability than Object-Oriented Systems.

In the view of developers, Service-Based System exhibits a high level of analyzability as compared to the Object-Oriented System.

The majority of the respondents believe that Object-Oriented Systems reduce the degree of testing complexities as compared with Service-Based Systems.

Rank based analysis
Developers believe that Service-Based Systems are better at modifiability, encapsulation and abstraction while Object-Oriented Systems demonstrate relatively a bit lower degree of complexity.

5.2 Qualitative Results

Eight experts are interviewed to gather insights into Service Orientation and Object Orientation w.r.t. maintainability.

One departmental head, one technical lead, four senior software engineers and two software engineers were interviewed in a semi-structured format. The experts were asked some general questions regarding work and knowledge in their respective disciplines. It was followed by showing them two functionally equivalent systems developed with two alternatives and discussing that in details. Afterward, they were asked about their experiences and opinions on various aspects of maintainability mainly revolving around a set of four essential elements modifiability, system complexity, analyzability, and stability. It is aided by some follow-up questions as well.

The expert interview ended with a discussion about the impact of quality attributes on software maintainability.

Demographic data of the experts is depicted in the following table.

Type of organization	**Number**
Product	4
Service	2
Research	2
Original background of interviewees	
Engineering	2
Machine learning	2
Information systems	1
Computer science	3
Age of interviewees	
Range of ages	
20-30 years old	1
30-40 years old	5
31-40 years old	2

5.2.1 Results from Expert Interviews

Question 1: In your opinion, What do you think about modifiability in Service-Based Systems and Object-Oriented Systems?
This question explores experts experiences with extensibility and changeability into Service-Based Systems and Object-Oriented Systems.

Responses from experts
In response to this question, Majority of the experts stated modifiability as highly essential quality attributes. Five out of eight experts admitted that Service-Based Systems are

better when it comes to extending already developed systems. Experts emphasized on the modularity of the services which allows creating an independent set of services and It further helps to reduce the cost of service implementation. Few of the experts highlighted that Service Orientation approach is much convenient when business requirements are frequently changing.

Question 2: In your experience, How do you see the size and complexities of Service-Based Systems and Object-Oriented Systems?
This question looks at exploring system size and complexities of the Service-Based Systems and Object-Oriented Systems.

Responses from experts
75% of the experts reported that Service-Based Systems are relatively complex as compared to Object-Oriented Systems. Experts considered Object-Orientation approach as broadly experienced and stressed mature tool support that exists for Object-Oriented Systems. Majorely, they raised their concerns over the structure and mechanism for implementing services and due to this reason stated Service Orientation as relatively complex.

Question 3: In your opinion, How do you see the analyzability of two alternatives, i.e. Service-Based Systems and Object-Oriented Systems?
This question tries to investigate the readability aspect of both the alternatives.

Responses from experts
Most of the experts expressed that service based system provides a better degree of analyzability. The primary reason for the preference was due to the independent service creation in a Service-Oriented approach. Experts addressed that independent service creation allows a greater extent of reusability which in turn improves the overall readability in the system.

Question 4: In your experience, How do you observe the stability of Service-Based Systems and Object-Oriented Systems?
This question explores the degree of stability concerning the Service-Based Systems and Object-Oriented Systems.

Majority of the experts indicate that Service-Based Systems promise a better degree of stability. Experts state that since services are independent it brings a lot of stability into the system. As per experts when there is a failure in service, it can be easily retrieved while in case of Object Orientation sometimes it may require cloning the whole system which leads to the memory overhead.

5.2.2 Discussion of key findings arising from the expert interviews

In this section, we will discuss the key interpretations and issues which are derived from the expert's interviews. In this process, Experts opinions are analyzed thoroughly and interpretations are made manually. These interpretations and issues are discussed one-by one.

Modifiability

As per experts, Service-Oriented System provides a better degree of modifiability since the amount of change in the code is comparatively less and straightforward as compared to Object-Oriented Systems.

Size and complexity
Experts opinions indicate that Object-Oriented Systems are less complex as compared to Service-Oriented Systems since Object-Orientation is equipped with a mature tool supports while on the other hand experts feel that services are comparatively complex to implement.

Analyzability
Experts suggest that Service-Oriented Systems posses a better degree of analyzability due to a lower degree of dependency among the modules while experts felt that Object-Oriented Systems have greater dependencies among modules.

Stability
Experts hints that Service-Oriented Systems provide better stability as compared to Object-Oriented Systems since they are better recoverable from system failures.

From experts interviews, three key issues were identified concerning the Service-Oriented approach.

Lack of tool support
Traditional technologies are willing to migrate in service orientation but they are facing a challenge with migration since Service Orientation lacks mature tool support for migrating the legacy applications.

Change in services
If a service is to be modified, then it impacts a large number of composite services. Generally, services are constrained to service level agreements and it affects profoundly to the service providers.

Grouping services
Another issue with Service Orientation is the grouping of services in a logical domain. Suitable grouping can allow a service provider to simplify the overall architecture. In this process number of components to be addressed are reduced.

5.3 Experimental Results

A group of eight developers contributes to the implementation of the experimental approach. Each of them is requested to perform development on a Service-Based System or Object-Oriented System. It consists of an exercise requiring a book search functionality in the already designed systems in respective approaches. This functionality allows the registered users to search for a book by its name.

Their work and efforts are tracked for the analysis of both paradigms.

Type of system	No. of developers	Exercise	Location
Service-Based System	4	Adding book search function into already developed systems	Germany
Object-Oriented System	4		

Table 5.6: Experiment information

The following figure shows the time taken by the developers in modifying a Service-Based System and Object-Oriented System. The modification consists of adding a search function in the existing SBS and Object-Oriented System.

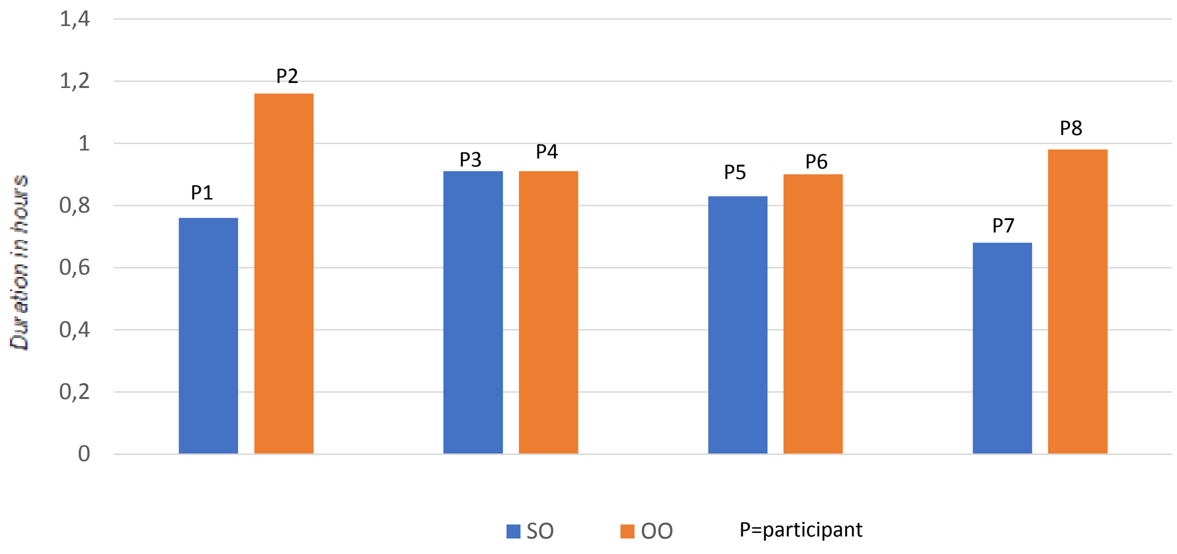


Figure 5.20: Development duration of the participants in SO and OO

Development efforts were measured from the lines of code written by the developers. Below figure displays the lines of code distribution in both of the systems.

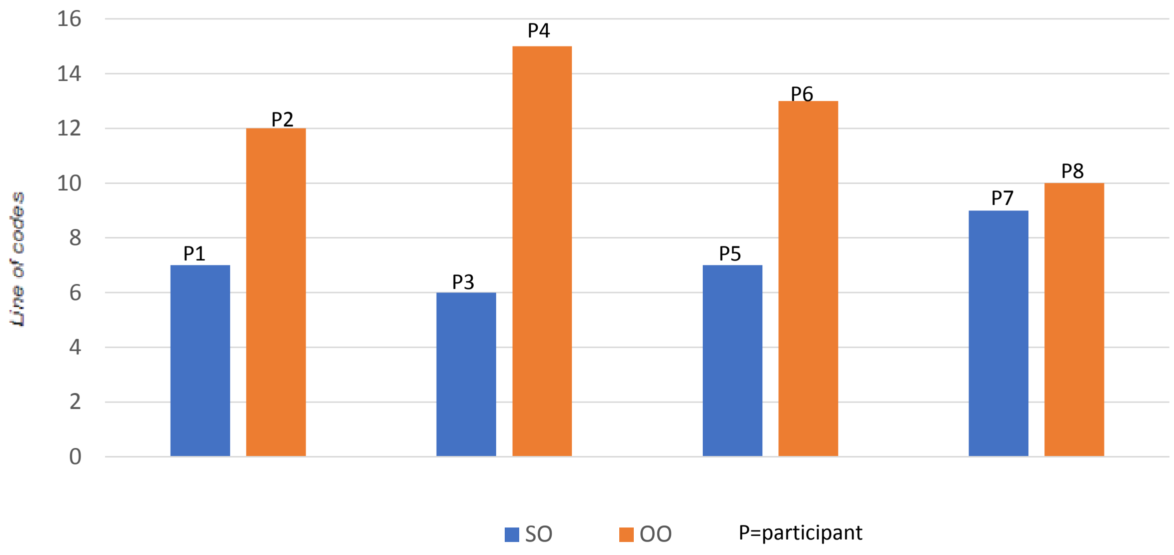


Figure 5.21: Development efforts of the participants in SO and OO

The figure 5.22 depicts the average development time devoted by the developers for the modifications in a Service-Based System and Object-Oriented System.
Experimental results indicate that Service-Based Systems are easier to extend as compared to Object-Oriented Systems and this provides another interpretation that they are less complex to modifications.

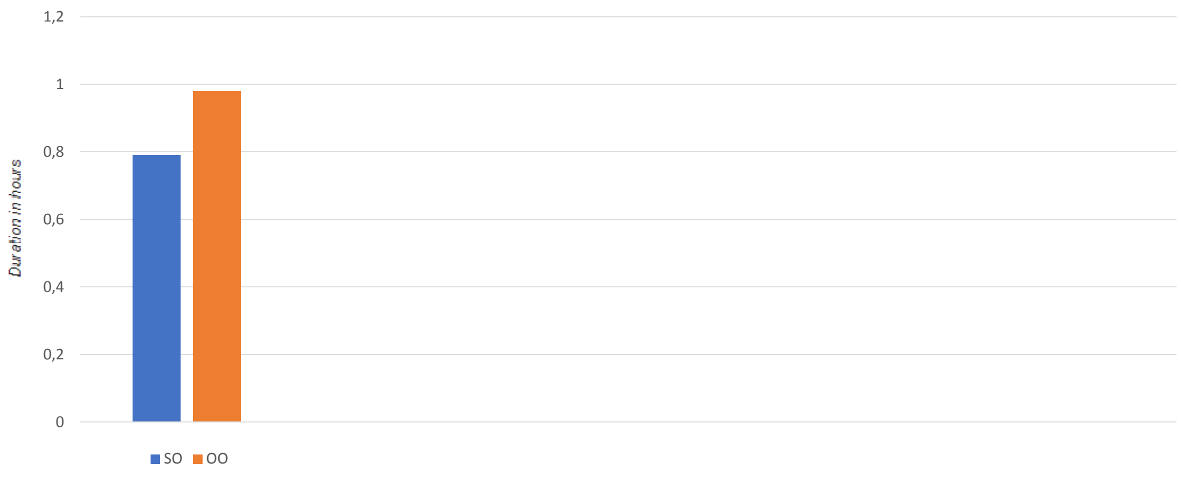


Figure 5.22: Comparative analysis of development time in SO and OO

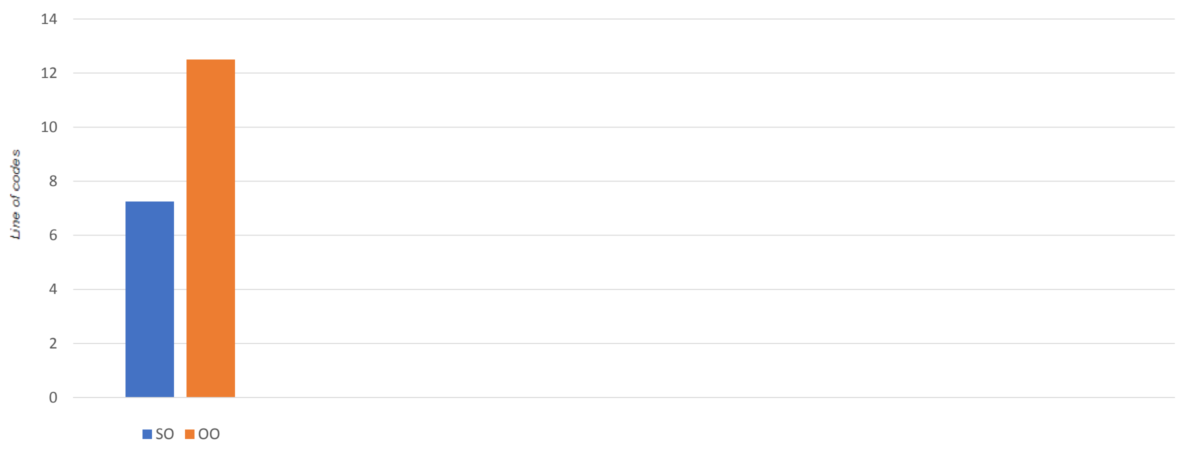


Figure 5.23: Comparative analysis of development efforts in SO and OO

Furthermore, development efforts analysis suggests that Service-Based Systems require comparatively fewer efforts than Object-Oriented Systems.

6 Limitations

While interpreting the results of this thesis, a number of limitations have to be considered in pursuance of applied research methodologies:

The literature review only covers coupling and cohesion quality attributes as affecting factors on maintainability. Other quality attributes can also have a significant impact on the overall maintainability of the software.

There are some limitations related to the software metrics. Firstly, the design and implementations are limited to the selection of the book and adding it to the shopping cart. Secondly, the implementation lacks data persistence. Such aspects can affect the structural attributes under evaluation.

Furthermore, Online BookStore(OBS) can be implemented in several ways using both Service Orientation and Object Orientation and different implementation may present different measures. A further limitation is that non-functional requirements were not taken into account while developing SO and OO system. As a consequence, non-functional requirements may have unknown effects on the systems.

Since software metric results are obtained by comparing a comparatively small service-based and Object-Oriented System, they may vary for significantly large systems. Moreover, the metrics are only evaluated for the design/implementation artifacts; Node libraries, Java libraries, etc. were not included.

Another limitation is that the Service-Based System is dependent on Object-Orientation approach where each service is implemented using OO development.

Like most-survey based studies and expert interviews, survey results and qualitative results from experts interviews may be subject to participants bias. Exclusively surveying the insights of developers and interviewing IT experts, who supposedly have useful information regarding the maintainability of Service-Based Systems and Object-Oriented Systems. We can build software of arbitrary quality with all three paradigms mentioned in the survey questionnaire. Therefore, the experience/quality of the developers is much more important than the chosen paradigm. Hence survey and experts interview result mirror the subjective affinity of developers and experts towards these paradigms.

The survey, experts interviews and experiment is conducted in one location(Germany). It limits the generalizability of our various approaches.

Furthermore, experimental results are collected from a small group of developers.

Developers relative experience may affect the work and efforts recorded from the experiment.

Nevertheless, despite such potential limitations, we think that our research approaches provide valuable insights on the maintainability of service-based and Object-Oriented System.

7 Conclusion

The purpose of this thesis is to provide limited generalizability on the maintainability of Service-Based System and Object-Oriented System. A case study was developed using two contrast approaches. Resulting implementations were measured using a set of six mutually consistent metrics.

7.0.1 Summary

The outcome of the software metrics suggests that 1) The Service-Oriented approach exhibit a lower degree of coupling than the Object-Oriented approach to a certain extent, 2) The Service-Oriented approach indicates a higher degree of cohesion compared to Object-Orientation.
Survey results imply that Service-Oriented System promises a better degree of modifiability, encapsulation and abstraction compared to Object-Orientation while object-orientation provides a reduced degree of testing and system complexity.
Experts interview indicates that Service-Oriented System presents a better degree of stability, analyzability and modifiability while Object-Oriented System tends to provide a lower degree of complexity.
Furthermore, experimental results denotes that Service-Oriented Systems provide a better degree of extensibility and changeability.

7.0.2 Future Research

The findings of this thesis contribute to the research on the maintainability of service-based and Object-Oriented System. Future research may utilize these results and provide further insights.

This case study is limited in two key quality attributes: coupling and cohesion. Additional quality attributes like size and complexity induce a significant impact on the overall maintainability of software systems. Therefore, further analysis of these attributes can provide more useful insights on maintainability.
Overall this thesis attempts to generalize the maintainability of a Service-Oriented and Object-Oriented System from the research perspective.
There is limited research exists that compares Service Orientation and Object Orientation. Thus, I hope that academics continue to explore maintainability from a research perspective and throw more light onto this essential quality attributes.

List of Figures

List of Tables

Bibliography

[1] T. DeMarco, *Controlling software projects: Management, measurement, and estimates*. Prentice Hall PTR, 1986.

[2] R. Land, "Measurements of software maintainability," in *Proceedings of ARTES Graduate Student Conference, ARTES*, 2002, pp. 1–7.

[3] L. Bass, *Software architecture in practice*. Pearson Education India, 2012.

[4] E. Arisholm, "Empirical assessment of the impact of structural properties on the changeability of object-oriented software," *Information and software technology*, vol. 48, no. 11, pp. 1046–1055, 2006.

[5] L. C. Briand, J. W. Daly and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[6] J. Al Dallal, "Object-oriented class maintainability prediction using internal quality attributes," *Information and Software Technology*, vol. 55, no. 11, pp. 2028–2048, 2013.

[7] "Business implications of soa." [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html

[8] "Service-oriented architecture: The business drivers for a new approach." [Online]. Available: http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:soa-ibmvision.pdf

[9] T. Erl, *Soa: principles of service design*. Prentice Hall Upper Saddle River, 2008, vol. 1.

[10] "Service architecture: Connections." [Online]. Available: https://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html

[11] "Service-oriented architecture: Soa collaborations." [Online]. Available: http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:soa-ibmvision.pdf

[12] "Service-orientation design principles." [Online]. Available: http://serviceorientation.com/serviceorientation/services

[13] "Soa principles : 5. service abstraction." [Online]. Available: https://www.slideshare.net/MohamedZakarya2/soa-principles-5-service-abstraction

[14] "Embolden low coupling with dependency injection." [Online]. Available: https://medium.com/@CStudio/embolden-low-coupling-with-dependency-injection-a1e6c1970872

[15] "Enterprise benefits on service oriented architecture – soa: Reusability." [Online]. Available: https://www.javacodegeeks.com/2013/03/enterprise-benefits-on-service-oriented-architecture-soa.html

[16] "Service reusability." [Online]. Available: http://serviceorientation.com/serviceorientation/service_reusability

[17] "Principle interrelationships and service layers." [Online]. Available: https://searchmicroservices.techtarget.com/tip/The-principles-of-service-orientation-part-6-of-6-Principle-interrelationships-and-service-layers

[18] "Capability composition." [Online]. Available: http://soapatterns.org/design_patterns/capability_composition

[19] "The service, the cloud the method: The connection points." [Online]. Available: http://2010.secrus.org/wp-content/uploads/download/Erl.pdf

[20] "Soa characteristic:enterprise-centric." [Online]. Available: http://2010.secrus.org/wp-content/uploads/download/Erl.pdf

[21] P. B. Crosby, *Quality is free: The art of making quality certain.* Signet, 1980.

[22] F. Losavio, L. Chirinos, N. Lévy and A. Ramdane-Cherif, "Quality characteristics for software architecture," *Journal of object Technology*, vol. 2, no. 2, pp. 133–150, 2003.

[23] P. Berander, L.-O. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö *et al.*, "Software quality attributes and trade-offs," *Blekinge Institute of Technology*, 2005.

[24] E. Iee, "Ieee standard glossary of software engineering terminology," 1990.

[25] B. Kumar, "A survey of key factors affecting software maintainability," in *Computing Sciences (ICCS), 2012 International Conference on.* IEEE, 2012, pp. 261–266.

[26] D. E. Peercy, "A software maintainability evaluation methodology," *IEEE Transactions on Software Engineering*, no. 4, pp. 343–351, 1981.

[27] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[28] S. K. Dubey and A. Rana, "Assessment of maintainability metrics for object-oriented software system," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 1–7, 2011.

[29] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on.* IEEE, 2012, pp. 306–315.

[30] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[31] M. H. Halstead, "Elements of software science," 1977.

[32] K. D. Welker and P. W. Oman, "Software maintainability metrics models in practice," *Crosstalk, Journal of Defense Software Engineering*, vol. 8, no. 11, pp. 19–23, 1995.

[33] J. Viljanen *et al.*, "Measuring software maintainability," 2015.

[34] M. Perepletchikov and C. Ryan, "A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software," *IEEE Transactions on software engineering*, vol. 37, no. 4, pp. 449–465, 2011.

[35] M. Perepletchikov, C. Ryan, K. Frampton and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian.* IEEE, 2007, pp. 329–340.

[36] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Transactions on software engineering*, vol. 20, no. 3, pp. 199–206, 1994.

[37] L. C. Briand, J. Daly, V. Porter and J. Wust, "A comprehensive empirical validation of design measures for object-oriented systems," in *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International.* IEEE, 1998, pp. 246–257.

[38] M. Alshayeb and W. Li, "An empirical validation of object-oriented metrics in two different iterative software processes," *IEEE Transactions on software engineering*, vol. 29, no. 11, pp. 1043–1049, 2003.

[39] D. P. Darcy, C. F. Kemerer, S. A. Slaughter and J. E. Tomayko, "The structural complexity of software an experimental test," *IEEE Transactions on Software Engineering*, vol. 31, no. 11, pp. 982–995, 2005.

[40] I. O. for Standardization and I. E. Commission, *Software Engineering–Product Quality: Quality model.* ISO/IEC, 2001, vol. 1.

[41] M. Perepletchikov, C. Ryan and K. Frampton, "Cohesion metrics for predicting maintainability of service-oriented software," in *Quality Software, 2007. QSIC'07. Seventh International Conference on.* IEEE, 2007, pp. 328–335.

[42] D. Rud, A. Schmietendorf and R. Dumke, "Product metrics for service-oriented infrastructures," *IWSM/MetriKon*, pp. 161–174, 2006.

[43] V. Saxena and S. Kumar, "Impact of coupling and cohesion in object-oriented technology," *Journal of Software Engineering and Applications*, vol. 5, no. 09, p. 671, 2012.

[44] L. Hui, H. Yu, J. Zhihong, H. Yuancan and H. Qiang, "High-cohesion and low-coupling integrative joint for space manipulator," in *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on.* IEEE, 2009, pp. 1463–1467.

[45] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI, pp. 259–262, 1995.

[46] Y. I. Mansour and S. H. Mustafa, "Assessing internal software quality attributes of the object-oriented and service-oriented software development paradigms: A comparative study," *Journal of Software Engineering and Applications*, vol. 4, no. 04, p. 244, 2011.

[47] G. Stubbings, "Service-orientation and object-orientation: Complementary design paradigms," *SPARK*, vol. 1, no. 1, 2010.

[48] M. Perepletchikov, C. Ryan and K. Frampton, "Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems".* Springer, 2005, pp. 431–441.

[49] V. B. Kampenes, B. Anda and T. Dybå, "Flexibility in research designs in empirical software engineering." in *EASE.* Citeseer, 2008.

[50] "Deduction induction." [Online]. Available: https://socialresearchmethods.net/kb/dedind.php

[51] P. Gandhi and P. K. Bhatia, "Optimization of object-oriented design using coupling metrics," *Optimization*, vol. 27, no. 10, 2011.

[52] "Online survey." [Online]. Available: https://www.smartsurvey.co.uk/

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 30.05.2018 Bhupendra Choudhary