A CBSE FRAMEWORK TO SUPPORT SYSTEM DEVELOPMENT WITH CHANGING REQUIREMENTS

By

Maj Raana Hafeez

00000280970

Supervisor

Assoc Prof Dr Tauseef Ahmed Rana

A thesis submitted to the Department of Computer Software Engineering Department, Military College of Signals (MCS), National University of Sciences and (NUST), Islamabad, Pakistan, in partial fulfillment of the requirements for the degree of MS in Software Engineering

September 2022

<u>THESIS ACCEPTANCE CERTIFICATE</u>

Certified that final copy of MS/~~MPhil~~ thesis written by ~~Mr~~/Ms **<u>Maj Raana Hafeez</u>**, Registration No **<u>00000280970</u>**, of Military College of Signals has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have also been incorporated in the said thesis.

Signature: ___________________________

Name of Supervisor: <u>Assoc Prof Dr. Tauseef Ahmed</u>

Date: ______________________________

Signature (HOD): ______________________

Date: ______________________________

Signature (Dean/Principal): ________________

Date: ______________________________

ABSTRACT

Over a period of time, Component-Based Software Engineering (CBSE) has emerged as a strong and widely accepted approach in the field of software engineering. This approach focuses on improving the drawbacks/ disadvantages that are commonly faced in the development of large scale software systems. In doing so, ***reusability*** of commercial off the shelf (COTS) components has become the backbone of CBSE. Based on this backbone, stand the key characteristics of cost reduction, enhanced software quality and faster time-to-market. However, different and multiple types of problems are being faced during development of a software system from COTS components. More or less, these problems are related to compatibility issues caused by the mismatches between the COTS components. Each of these problems is a domain of study in itself and much research is being carried out for their solutions. This starts from searching components from online component repositories. When found, analyzing the component for making it compatible with own system being developed, testing the component and finally integrating into system.

Although complete development lifecycle exists for component based development and solutions are being proposed to the practical problems faced during and after development, a framework lacks which can serve as a guideline for developers by following which they can overcome major problems especially faced when carrying out a change in component based systems. This thesis aims to study and propose a framework for component based development cycle, able to incorporate a change at any given time, focusing to deal with selection and mismatch resolution issues effectively so as to reduce time consumed. The proposed framework also gives a guideline for developers to search and integrate required components in their systems with constantly changing demands and requirements. This study will facilitate developers to carry out change in a running component based system. All necessary steps required will be available as guideline to developers in logical order to avoid extensive searches from online repositories and component mismatch issues and ease the testing and integration process. The scope of the thesis does not include aspects of cost and non-functional requirements. Cost and non-functional requirements are important aspects of any project. Therefore, these two factors will be a part of future enhancement of the proposed framework.

ACKNOWLEDGEMENTS

I pay my gratitude to Allah almighty for His blessings and guidance without which I would not have been able to complete my degree and thesis. I would also like to express special thanks to my supervisor Assoc Prof Dr. Tauseef Ahmad Rana for guiding and encouraging me to complete my thesis. I would also like to thank committee members Assoc Prof Dr. Fahim Arif, Assoc Prof Dr. Ihtesham Ul Islam and Asst Prof Muhammad Suhail for guiding me during thesis work. Their timely and efficient contributions helped me shape the thesis into its final form and I express my gratefulness for their sincere supervision all the way. I am also thankful to Department of Software Engineering, MCS, and the teachers for providing me with an academic base, which enabled me to complete this thesis. Finally, I would like to express my gratitude to the people who have been supporting me not only during thesis but throughout the MS degree; my dear mother, supportive husband, siblings and my naughty children. I owe it to all of them.

DEDICATION

Dedicated to my father, Abdul Hafeez, whose presence would have completed my joy of every accomplishment, and my daughter Fizza, who was blessed to me at start of my degree, whose eternal love drove me through the entire journey of my MS degree

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Overview

Component-Based Software Engineering (CBSE), over a period of time, has emerged as a strong and widely accepted approach in the field of information technology. This approach focuses on improving the drawbacks/ disadvantages that are commonly faced in the development of large scale software systems. In doing so, ***reusability*** of commercial off the shelf (COTS) components has become the backbone of CBSE. Based on this backbone, stand the key characteristics of cost reduction, enhanced software quality and faster time-to-market. However, besides these advantages, a lot of mature effort is required for not only development of these component based systems, but also for their maintenance. With the increase in competition in software industry, much of focus has been on reduction of time to market and cost, but along the way complexity in type of software has also increased manifolds. So is the type of components. Large component repositories are available online. But due to complex nature of components, search and selection of components from these online repositories are a vast area of research. Basic techniques like categorization, keyword searches or even a full text search are usually insufficient. Tools and specialized search engines like Agora [1] are being developed to carry out exhaustive, yet efficient search from large repositories.

Components send messages to each other for interaction. These messages depend on the signature of the components. To interact smoothly, components should be able to understand each other effectively, meaning they should be compatible. Their signatures should not mismatch. This aspect is often neglected resulting in wastage of time and effort. All the efforts spend in searching and selecting component also goes waste. Therefore, it is strongly recommended to analyze the signatures of the component that have to interact with each other carefully before starting the process of adaptability of components. Usually compatibility problems are caused by the semantic and syntax mismatches between the COTS components. For integrating COTS components to function smoothly, these mismatches are important to resolve.

With evolution of software applications and increase in complexity of nature of software nowadays, dynamic properties of systems are important to be analyzed. These properties include dynamic reconfiguration, as an example, or flexibility of components etc.   Hence it now requires a strategy that provides an opportunity for a software developer to shortlist the component that either fully or moderately provides the preferred functionalities. Complex nature of systems, constantly changing needs of customers, up gradation of versions make change inevitable. Flexibility is mainly essential for the process of component selection as it increases the possibility to find strongly matched components. Handling a change in component based systems becomes more challenging due to strong inter dependency of components on each other. Change in one component can cause ripple effects that can cause in overall failure of the system. Therefore, change management in component based system depends on careful analysis of overall assembly of components and their interactions.

1.2 Problem Domain

With the three critical problem domains discussed above, changing requirements, search and selection of COTS components and resolution of mismatch issues, the developer has to study each domain separately to deal with any kind of change required in the system. There is a logical order to carry out any task in any discipline of study; same is with carrying out change in a component based system. Activities performed in one phase always affect the subsequent phases. A phase carried out properly can ease the tasks to be performed in subsequently. There is a lack of this logical order that can help a developer to deal with the three problem areas systematically. It is to emphasize that these three problem areas are deeply related to each other when it comes to incorporate a change in any component. Hence, there is a lack of well-defined framework or model, following which can help the developer to carry out a change in running system smoothly.

1.3 Problem Statement

CBSE is facing lot of issues that impacts on the quality of component due to lack of a framework that will provide developers a guideline in developing component based software with varying requirements. This research aims not at finding solution to any of the problem domains as discussed earlier, but to study and find a logical order in which these domain areas need to be addressed, so that developers can incorporate a change without omitting important steps that can cause further delay.

This thesis aims to study and propose a framework for component based development cycle, able to incorporate a change at any given time, focusing to deal with selection and mismatch resolution issues effectively so as to reduce time consumed.

1.4 Research Questions

The purpose of this thesis is to study and find solution to the following questions:

1. What will be the component selection criteria from large repositories available on internet
2. What will be the mechanism to resolve the mismatch issues from selected components to make them compatible with a system
3. How will the component be inserted into the system after removal of mismatches
4. How will the components be kept in repositories
5. What will be a complete framework that can be followed by developers

1.5 Objectives

The main objectives of the thesis are:

1. Comparative analysis of already existing frameworks/models for identification/ selection of components.
2. Identify issues faced by developers to identify and select required components in systems with constantly changing demands and requirements.
3. Propose a framework for component selection with dynamic requirements and resolve the mismatch issues.
4. Propose a framework that encompasses the all the steps involved in identifying change in system's behavior, analyzing it, addressing it with component change and finally replacing a particular component responsible for the change.

1.6 Areas of Application

It can be implemented in all areas requiring separation of concern, mainly categorized into following.

1. Web Services
2. Service oriented architecture
3. Event driven architecture

1.7 Thesis layout

The layout adopted for this thesis is as follows:

1. **Chapter 1**: In chapter 1, an overview of the CBSE domain, problem domain, problem statement, research questions and objectives have been described briefly.
2. **Chapter 2**: In chapter 2, background of the topic has been described. Problems of changing requirements, selection of components and resolution of mismatch issues have been discussed.
3. **Chapter 3**: This chapter contains literature review of work already done in the field of changing requirements in products developed through CBSE.
4. **Chapter 4**: This chapter describes the proposed model briefly. It describes the basic flow of events in the proposed framework.
5. **Chapter 5**: In chapter 5, the proposed framework has been elaborated in detail with the help of examples.
6. **Chapter 6**: In this chapter the proposed framework has been evaluated with a test case.
7. **Chapter 7**: In this chapter, the thesis has been concluded and future scope of the topic has been discussed.

1.8 Conclusion

The chapter briefly describes the basic structure of the proposed research work. The issues in component based development have been touched upon, mainly different problem domains in the domain of component based development. The problem that needs to be addressed has been narrowed down, and the way has been identified as how to find a solution for the problem.

Chapter 2

BACKGROUND

2.1 Overview

With success, come challenges as well. CBSE has experienced the same. After being widely adopted, certain challenges have appeared that require constant solutions. Searching a suitable component for a project, from large repositories is a critical challenge. Moreover, writing code for making the components interoperable with rest of the components in project is another time consuming task. Focusing on these two problems, in this chapter we will review the advancements done in field of CBSE, so as to highlight areas that need attention.

2.2 Software component

A software component is basically a software unit with a well-defined interface and explicitly specified dependencies. A software component can be as small as a block of reusable code, or it can be as big as an entire application [2]

2.3 Need for CBSE

In this competitive era of fast changing IT industry, increasing need of digital systems, growing complex nature of software, building every software from scratch is not a suitable option any more. This is an age of rapid and agile development focusing on characteristics like ***reusability*** and ***scalability***. That is why CBSE is now an emerging and widespread approach used for developing component based systems.

With CBSE, issue of **duplication of effort** has been curtailed. An effort made once, can easily be reused. Moreover, the structure of a component based system is such that it places common functions at one place, thus making code more **understandable** and easy to modify. Often a need was felt to create a functionality based on existing functionalities. CBSE allows creating a new component by **extending** two or more components. Hence, without duplicating effort, we get a new behavior based on existing behaviors. CBSE also fulfills the need of **encapsulation**. A complete service can be offered without revealing details. Functionality, like a clock, is often required to operate in the same manner under different environments. So the need for context free development is also addressed by CBSE.

In short, multiple needs gave birth to CBSE approach, which is running successfully today. But it is to emphasize, that while enjoying the benefits of CBSE, due focus has to be given to the problems that have risen due to following of this approach. Timely solutions will further pave the way of success of CBSE.

2.4 Advantages of CBSE

Because of the architecture, component based development offer those advantages that cannot be achieved through traditional software development. The flexibility offered by a component based design i.e., arranging components into a manageable and well suited pattern for an organizational needs, has made CBSE the most adopted approach in a competitive environment, where more output is expected in less time. Few major advantages of CBSE approach are given below:

1. The major advantage of CBSE is reusability. This is like prime signature of this approach.
2. Component based systems are easy to maintain, due to separation of concerns and encapsulation.
3. Systems are developed in less time.
4. Due to reusability, cost of development is significantly reduced.
5. Better understanding because of the structure, similar functionalities grouped together.
6. Separation of concerns provides a better modeling of the overall system.
7. Works independently.
8. Easy deployment, by just replacing a component does not affect the whole system.

2.5 Sources of Components

Software components can be purchased from different sources. It usually depends on the type of requirement that an organization is facing.

1. When an organization is unable to carry out in house development for any specific task, it relies on different **IT services firms** for the require component.
2. For a generic task, **packaged software producers** are preferred.
3. Complete systems are purchased from **enterprise solution vendors**, which cross functional boundaries.
4. **Cloud computing** is utilized when an instant access to an application is required.

5. When cost is high for a generic purpose task, components are purchased as **open source software.**
6. For building from scratch, **in house developers** are utilized.

2.6 Software Repositories

Different component repositories help the programmers to find a suitable component required. The meta data available in these repositories also ease the developers to understand the composition of the components and write necessary code to modify/ adapt the components. These repositories are generally composed of three different parts; a repository containing different components physically, a mechanism well defined to carry out necessary search and retrieval of the components, and an interface for user interaction [3]. They provide developers with a platform where various pieces of source code can be stored by different providers and can be accessed by various solution seekers.

2.7 Online Software Components Repositories

Nowadays, many component repositories are available online. Each repository has certain attractions for certain set of users. It is up to different users and organizations to select their online repositories as per their technical/ non-technical needs. Most commonly used software repositories are:

Language	**Repository**
Java	Maven
.NET	NuGet
Php	PECL
Python	PyPI
TeX, LaTeX	CTAN

Table 1: Online Component Repositories

2.8 Benefits of Online Software Components Repositories

1. **Effective Management**. The components, their versions and their metadata is managed by owner of repository.
2. **Ease of Access**. Irrespective of an organization's location, current and up to date information/ components are readily available.
3. **Increased Responsiveness**. With readily available components, it becomes easier for developers to find solutions to their day to day programming tasks; either it's an application as component, like online payment system, or a small solution like a calendar or a chart.
4. **Productivity of an Organization**. Increased responsiveness helps organizations meet their deadlines timely and respond to customer demands efficiently. Therefore, productivity of organization increases and helps maintain a good name in competition industry.
5. **Application of Expertise of Domain Specialists**. Components written by respective domain experts, not belonging to own organization are a great asset that any organization can use and shortlist for future use.
6. **Conversion Issues**. As metadata of components is available, conversion of reference values is done in minimal time.

2.9 Component Based Models

Multiple component based software development lifecycle have been created by various researchers and practitioners. Some of such very well-known models are briefly described below:

2.9.1 V Model

The V model is an enhancement of traditional waterfall model, but more flexible in nature than waterfall model as it is intended for component development. There is a separate system for component search and evaluation which provides input to selection phase. Testing is a key feature of this model and is more emphasized as in waterfall. Test plan and test cases are developed before start of implementation/ coding phase.

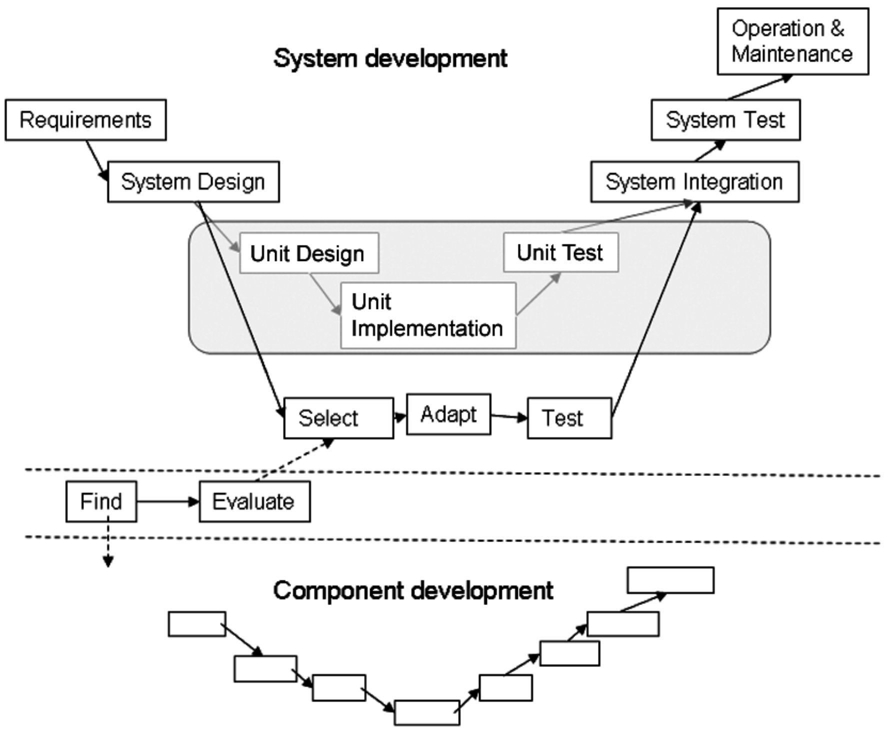


Figure 1: The V Model

2.9.2 Y Model

In Y model, overlapping and iteration is facilitated where it is required. Therefore, in the development process, a lot of changes are involved, hence instability so created have to be dealt with. Some new phases were introduced in Y model including domain engineering, frame working and assembly.

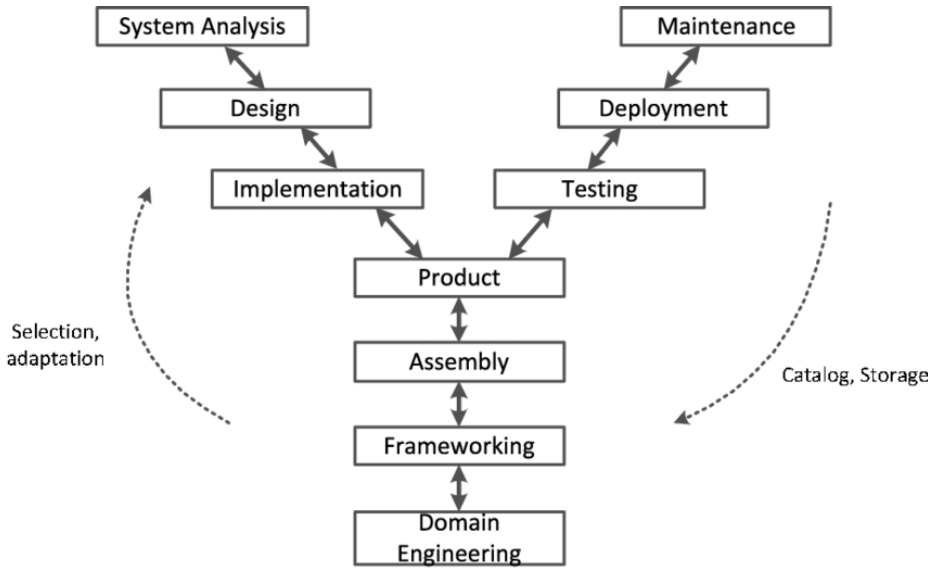


Figure 2: The Y Model

2.9.3 W Model

The W Model consists of two V models, one for component and one for system lifecycle. There are two phases in component lifecycle, design and deployment. Software components are identified in design phase, then designed and constructed accordingly. Finally they all domain specific components are put into a component repository. In deployment phase, product is deployed successfully.

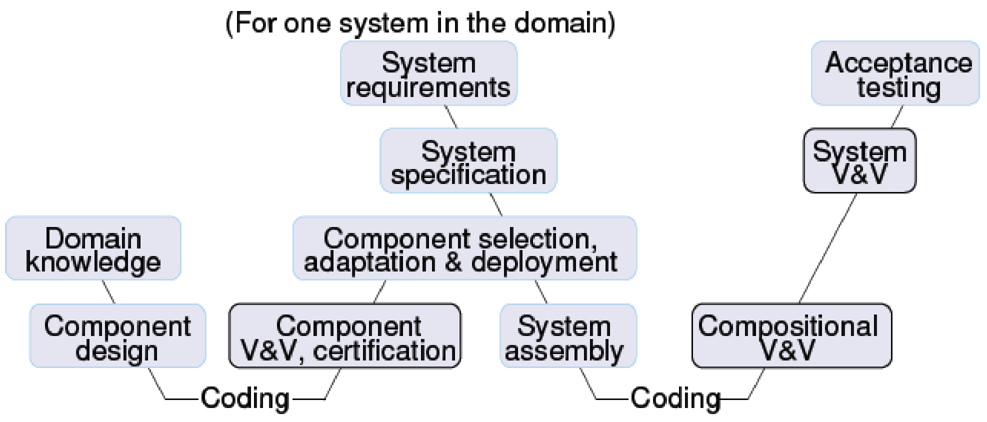


Figure 3: The W Model

2.9.4 The X Model

The X model separates component development process from component based software development. It consists of four sub cycles if development as shown in figure 4:

1. Development for reuse
2. Development after modification
3. Development without modification
4. Component based software development

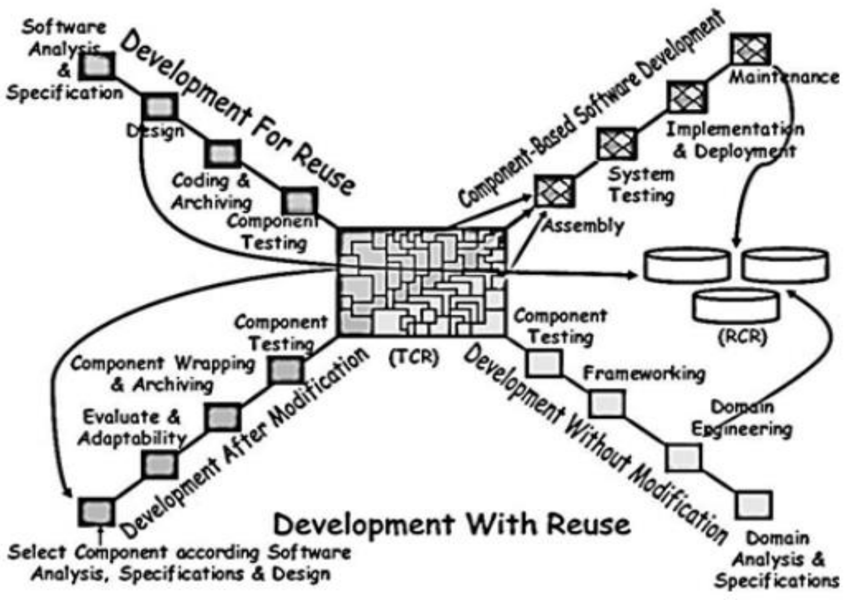

Software Analysis & Specification
Design
Coding & Archiving
Component Testing
Development For Reuse
Component-Based Software Development
Assembly
System Testing
Implementation & Deployment
Maintenance
(RCR)
(TCR)
Component Testing
Component Wrapping & Archiving
Evaluate & Adaptability
Development After Modification
Select Component according Software Analysis, Specifications & Design
Development With Reuse
Component Testing
Frameworking
Domain Engineering
Development Without Modification
Domain Analysis & Specifications

Figure 4: The X Model

2.9.5 Elicit Model

Development with reuse is the main concept of the Elite model. Components are assembled through selection, development and reuse. Elicit model supports outsourcing of components.

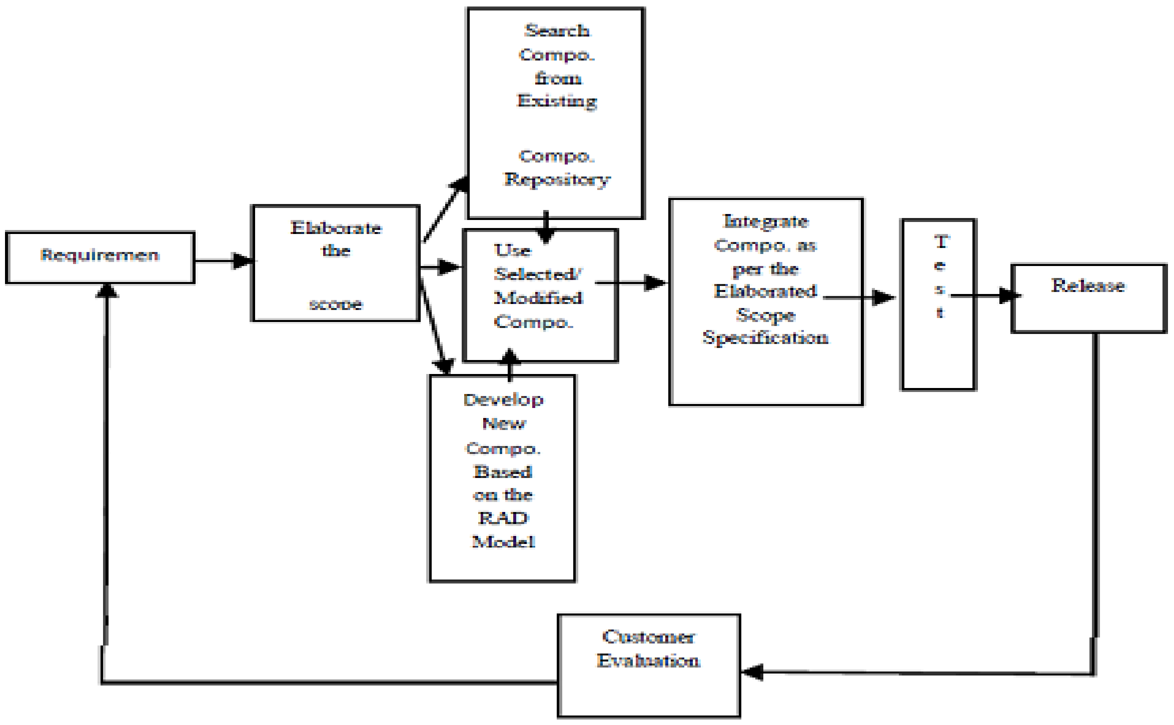

Requiremen
Elaborate the scope
Search Compo. from Existing Compo. Repository
Use Selected/ Modified Compo.
Develop New Compo. Based on the RAD Model
Integrate Compo. as per the Elaborated Scope Specification
T e s t
Release
Customer Evaluation

Figure 5: Elicit Model

2.10 Component Based Development Issues

Along with advantages, there are many challenges that are faced when CBSE approach is adopted. More appropriately, these are taken as tradeoffs of the numerous benefits offered by CBSE. This thesis considers two major issues, ***selection of components from an online repository*** and ***resolution of mismatches between components***, which are discussed in subsequent paragraphs. Some other concerns are as follows:

1. **Security**. As the source code of components is not known, it offers security concerns to its users, as there can be unknown uses of components.
2. **Compatibility**. Components may not be compatible with an organization's middleware technologies. So a lot of effort may have to be put in to make it compatible with own project being developed.
3. **Testing**. Testing is difficult as implementation is unknown. So testing of components with maximum possible test cases is not possible.
4. **Trusted components.** CBSE is mainly about black box development. Details of implementations unknown. Therefore, besides security concerns trustworthiness of a component is a common and an important aspect.
5. **Requirement Management**. Reusability is the basic concept behind component based development. In such a scenario, requirement engineering becomes a complex task as the best suited component may not fulfill all the requirements. Normally, the best component is closest to the requirements, may lack one or more functionality.
6. **Selection of Components.** Selection of components has always been one of the important tasks in developing component based products. It is not only tedious but also involves a lot of challenges for the developers. By challenges, we mean that as software has grown complex in nature, so is the criteria to define them in order to carry out search. Therefore, search involves multiple criteria as per the requirement of the project. Existing study shows that different types of COTS selection methods have been introduced. A few of them are describes briefly here:
   a. **Off-The-Shelf Option (OTSO)**. Two main techniques are used in this method; Goal Question Metric (GQM) approach to evaluate components against given criteria and Analytic Hierarchy Process (AHP) for selecting the best component.
   b. **Procurement-Oriented Requirement Engineering (PORE)**. This approach works in three phases. COTS vendor provides product information and customer requirements in Phase I. Vendors-led demonstration provides product information

and customer requirements in Phase II. Customer-led product exploration provides product information and customer requirements in Phase III.

c. **Social Technical Approach to COTS Evaluation (STACE)**. Practically, it has been observed that non-technical issues like human, social and organizational characteristics play a vital role in COTS selection. In this approach, high-level requirements are decomposed into the small and measurable attributes and component evaluation and selection are based on social as well as technical criteria.

d. **COTS Aware Requirements Engineering (CARE)**. This approach includes functional, non-functional and architectural effectiveness while forming evaluation criteria. Major steps involved in this approach are:
   i. Define goals
   ii. Match goals
   iii. Rank components
   iv. Negotiate changes
   v. Select components

e. **Storyboard**. Customer requirements are the key feature of this approach. Various steps of this approach are:
   i. Gather customer requirements
   ii. Identify components based on customer requirements
   iii. Develop use cases
   iv. Evaluation of components

7. **Component Mismatches**. As mentioned above, best suited component may not fulfill all requirements. Further to this problem, it is not necessary that the best component will function optimally with the rest of the components in overall system. So there is a need to analyze and evaluate the interaction of component with rest of the system.

2.11 Conclusion

In this chapter, we have reviewed CSBE from its inception and have cycled through various phases of this approach. Apart from development needs that CBSE satisfies, advantages offered, we have reviewed different models proposed to meet those development needs and take full advantages of the benefits offered by this approach. Study has shown that most of the problems faced by developers during development cycle are solved during runtime and not documented. This is a major reason that evidence of issues and particularly their solutions is lacking. Due to lack of evidence, some of these issues have become major problems, like component selection and component mismatches. Multiple efforts have been put in to propose solutions to these problems. Hence, these problems are no longer a sub domain of CBSE and have evolved as a domain in them.

Chapter 3

LITERATURE REVIEW

3.1 Overview

CBSE offers a number of advantages as well as challenges. These challenges have been addressed by different researchers for the past few years. The research is still in process as the challenges are dynamic in nature due to black box nature of COTS. Many problems faced by the developers are run time in nature. Many of the solutions adopted by the developers were based on scenario and resolved at the spot by experience and were not documented. Developers have been using their past experiences with components to resolve future issues. That is why challenges of CBSE have always been a vast area of research. In this chapter, already available research on various models and issues of CBSE has been studied. A few of those papers have been discussed here. They have been organized under three main domains; development, identification and selection and non-functional requirements.

3.2 Development Frameworks

3.2.1 Self-adaptive Model

Tang and Liu have proposed a self-adaptive software architecture model [4] to support integration of heterogeneous COTS components by use of connectors. The architecture of connector has been described in detail which is the key player of the proposed model. They have investigated the problem of identification of mismatches between COTS components including semantic, data format and behavioral mismatches. Finally, they resolved various components mismatches by implementing message interceptors, data buffers and data assemblers, all of them forming part of a connector. This paper provides a good and comprehensive framework for integration of components. Only one phase of searching and retrieval of component have not been discussed in this paper.

3.2.2 Model-Driven Software Development

In this paper Alrubaee et al [5] combine the concepts of CBSE and model driven software development (MDD) and propose a new software development process model termed as Component based model driven software development (CompoMDD). The model is composed of two main activities. First is component development, in which new components are developed from scratch. It includes phases of model specification and implementation for both reusable and non-reusable components. Second activity is system development which comprises of requirements elicitation, project planning, system analysis, system decomposition, component conceptual modeling, components searching and selection, adaptation, components integration, system implementation, system test and deployment. MDD is embedded in both types of development. This paper differs with our thesis in one aspect only i.e., components are developed, COTS products are not utilized. Phases of adaptation, integration and testing are very strong of the proposed model.

3.2.3 Framework for Adaptable Components

Customizable system can be adapted during run time as per varying needs of the system to function properly. Rosa et al, in their paper [6] have proposed a general framework for such customizable systems that automatically decides the reason for adapting the system and the changes required to be done to carry out the adaptation. The proposed model defines the behavior of the system as a final policy that spells out the goals of the system and the factors against which measurement needs to done. All the information available, regarding the components wil help in making the required decision. The effect of any modification on the rest of the system is also taken into account. Two phases are involved in this technique, online and offline. In offline phase, goals are defines and rules are set according to which adaptation can take place. In online phase, these rules are applied for any change to occur. This paper only discusses customizable components, whose parameters can be set at any time during execution of the system. That is how change is applied to the system. Whereas we want to find a solution in which change has to be applied by changing a component.

3.2.4 Development Using Component Oriented Programming

Shukla and Marwala [9] have illustrated programming concepts for component based systems in their paper. First they have described five steps of COTS based development lifecycle; requirements analysis, software architecture selection and creation, component selection, integration and component-based system testing. They have compared this lifecycle with the traditional waterfall model and have identified the similarities and differences between two lifecycles. Then major component frameworks, CORBA, COM and EJB have been discussed. To clarify the concepts of component oriented programming, a sample application was also illustrated. Different advantages as well as challenges of COTS development were presented. According to authors, component based development should be adopted, once all the challenges and risks associated with it are fully known and organization has the required expertise to overcome those challenges. Overall a comprehensive paper for beginners, but does not touch critical challenges that are being addressed in our thesis work.

3.2.5 Behavioral Models

Components interact with each other through connectors. During this interaction, a lot of data is generated and passed from component to component. Lu et al in their paper [11] suggest that if this data can be collected during the execution phase, then its analysis can lead us to understand the behavioral model of the component based system. The authors define the behavioral model as the internal behavior of each component along with the interactions of various components amongst themselves. They have proposed a two-step approach to discover behavioral model of a component based system. First is component discovery, in which components are discovered along with their organization. Second is interaction discovery, the interactions between the components is studied. To prove their approach, authors have used process mining toolkit ProM. A major assumption of the proposed approach is that documents generated during each phase of the development lifecycle are complete and we are in complete picture of how the components have been organized.

3.2.6 A Survey of Component-Based Life Cycle Models

Negi et al [15] have compared various models of component based software engineering in their paper. They are of the view that each model has its own advantages and disadvantages. It depends on organizational and project needs to decide which model to follow. Before comparing development, they touch important attributes of a component in a glance. They also briefly describe benefits as well as difficulties of CBSE. Different component based lifecycle models compared by authors are the V, Y, W, X, Knot, New Era, Elicit and Elite Plus. The V model is an enhancement of traditional waterfall model where more emphasis is given on testing. In Y model new phases like domain engineering are introduced. Two V models join to form W model. It comprises of two phases, component design and component development. The X model focuses on three types of scenarios, development for reuse, development after modification, development without modification and component based software development. Risk analysis is the key feature of Knot model. New Era model introduces selection and customization concept for components. Elicit model also focuses on selection and development of component with reusability as key factor. This survey paper provides a solid base for researchers in formulating new models/ frameworks.

3.2.7 Development through MVC component-based approach

In this paper, Sastypratiwi and Yulianti [16] propose that components can used in different forms. They aim to produce a component, using propose a Model-View Controller (MVC) model, in a form that can be used in website development as well besides in application. They first implement use case diagram to show interaction of user with the system. Then they use activity diagram to show how activities of the system take place and what the flow of control between user and system is. In last, they show the component arrangement in overall system and the interactions among various components through component diagram. They have used CodeIgniter to develop an MVC based vehicle testing application in PHP. The component so produced is placed in a library. This component can then be easily called by PHP developers in any project. The model is language specific, hence a big limitation. It can not be used by researchers in generic purposes.

3.2.8 Architectural Description Language

In this paper [19], Selic proposes a component based architecture to deal with dynamic nature of changing structures of today's modern applications. The author first explains how architectural description languages (ADLs) patterns are used to make the design of the component based system understandable and to be communicated to others. In this paper two architectural design patterns have been proposed, dynamic part pattern and the dynamic role pattern. In dynamic part pattern, it is proposed that the application will itself control the creation and termination of a dynamic part, meaning creation of dynamic objects during runtime. The relationship of these dynamic objects with other components, meaning their coupling, is also dynamically created through connectors. As per author, these two design patterns are sufficient to deal with any changes. They also avoid the implementation of unnecessary writing of code. To describe the working of patterns a running example of dating application was illustrated. A simple component-connector model was implemented. A ClientManager component is responsible for creation and deletion of all client instances dynamically. RelationshipManager is responsible for creation of relationship between different clients. Hence both patterns are well describes by two main components. This paper also utilizes the implementation details, therefore, it is not applicable in the COTS scenario.

3.3 Component Identification and Development Frameworks

3.3.1 Intelligent Agents

Abraham and Aguiler [7] have adopted an intelligent approach and have designed a model to select component from large online repositories. Components are usually placed in different repositories that are found online. Each such component is defined in form of an XML file through which the selection is done. All important information like functionality and description are defined in this file. This file serves as a component profile. Each component may have a sub profile depending upon the number of environments under which it can operate. One extra field is also stored to be used by this algorithm; this field is called “pheromone”, which is a concept taken from collective intelligence theory. Whenever a component is tested for performance, a pheromone is added to its profile. This pheromone trail is then used further for component selection. This paper proposes an algorithm for component selection which is not only mathematically strong but also light weight to handle. Again, component selection is just one phase of component based development. This paper offers solution for just one phase of the solution that we are seeking.

3.3.2 A Research Review

Gholamshahi and Hasheminejad [8] have studied different component identification and selection methods in their survey paper. They have presented various strengths and weaknesses of the studied methods. For component identification, two major techniques, clustering and heuristic approach have been illustrated. Apart from these two techniques, methods of graph partitioning, artificial neural network formal concept analysis, CRUD and affinity analysis have also been discussed. For component selection, traditional methods of weighted sum method (WSM) and analytic hierarchy process (AHP) have been discussed. Main focus for selection process is on the five step process proposed by

Six Sigma for component selection. Component identification and selection is only the first phase of complete component development lifecycle that has been focused in this paper.

3.3.3 Criteria Catalog

Carvallo, Franch and Quer have organized selection criteria into a criteria catalog (CC) [17]. A CC was built for a scope which either includes a domain (workflow systems, mail servers, antivirus tools, and so on) or a category of domains (communication infrastructure, collaboration software, and so on). With the implementation of CC, authors have achieved a hierarchical tree-like structure which serves the purpose of selection criteria. Authors have divided the process of software selection into four sub processes, requirement specification, availability of required software packages, and evaluation of available packages in accordance with the requirements and finally, based on evaluation selection of the package that best meets the requirements. They first define a basic CC and then further enhance it by considering different problem scenarios faced in component selection. This paper presents a strong and a quick mechanism for selecting components as per organization's need but doesn't address the subsequent phases of development.

3.3.4 Identifying Through APIs

When compared with implementation at object oriented class level, Shatnawi et al [22] found that reusability is more often practiced at component level due to provision of required interfaces. However, there are various classes and methods of APIs that can be reused to solve a particular problem. In his approach, the author uses the interactions between client applications and the targeted API to identify specific software components that can be reused. Multiple scenarios are executed and groups of methods that appear together many times are packaged. Graph based clustering algorithm is used to identify such groups of methods. DaCapo benchmark has been used for evaluation of their approach and has received a precision of 98%. This approach goes down to implementation details. Therefore it is only possible in case of open source components. In case of COTS components, this approach cannot be applied.

3.3.5 IFSOM

Bali, Bali and Madan propose a framework [21], intuitionistic fuzzy set and optimization model (IFSOM) for evaluation and selection of software components. The framework consists of two phases. In the first phase, multiple qualitative criteria are used for evaluation of COTS vendors. Techniques of intuitionistic fuzzy set (IFS) and technique for order preference by similarity to an ideal solution (TOPSIS) have been used in first phase. These vendors are then assigned different ranks depending on their evaluation. The ranks are further optimized in phase two. The model used for optimization uses constraints of reliability, cost, and delivery time. Hence, best components are selected from vendors with highest ranks and also under constraints of reliability, cost, and delivery time. This model uses components both in built and bought from vendors. A case study was carried out where a project was undertaken by a company in which they have to develop three modules, front office, back office and finance. Four vendors were selected (A1 to A4). The component built in house was termed as A5. The proposed two phased model was applied step by step. The proposed model is a tedious process involving mathematical computation for just one phase only, whereas we aim to propose a complete model from selection to delivery with minimum possible time.

3.4 Models Based on Non Functional Requirements

3.4.1 Reusability: Emerging Trends

Capilla et al, in their paper [10] discuss various forms of software reusability. They have started from early forms of reuse of source code and domain analysis. Various cost models used earlier for component cost estimation also discussed. According to authors with advancement of technologies and current need in industry and application domains, focus should now be more towards new trends of product lines, features and context analysis. Moreover, availability of open source software, services and micro services are latest forms of reuse. As per the authors, software reusability can never cease to exist, instead new merging technologies to support reusability are harnessing day by day, and component based development being one of them. Authors have discussed various domains like automotive, space, home appliances etc where software reusability is widely practiced. They have concluded that to cater for the complex requirements of complex software, reusability will always be a key element to handle such complexity. The paper comprehensively focuses on concept of reusability as whole, but does not touch upon any implementation details.

3.4.2 Collaboration between COTS Stakeholders

To manage change in COTS based systems, Ravi, Hadar and Levy [12] have focused first on identifying risks related to COTS based system that can hinder implementation of change. Then they found the means to mitigate those risks. The proposed methods for mitigation are based on advanced information system development (ISD) methodologies focused on human, collaborative and knowledge aspects. At last they focus on knowledge gaps between various stakeholders whenever a change request (CR) is initiated. These gaps can be a result of misunderstanding both the technical and domain knowledge of the system. As a case study, on appearance of a CR, various stakeholders were interviewed. They include the senior management, developers, testers and customers as well. The knowledge of the application domain was different for each role, which eventually lead to knowledge gaps. When this was addressed, the CR became easy to apply. This paper focuses on knowledge base of different stakeholders, the level of communication amongst them. It implies that with all stakeholders on board, on same page, CR becomes easy to handle.

3.4.3 Security in Component-Based Software Development

In this paper [13] Jha and Mishra argue that security for component based applications have never been addressed fully. Security has a great impact on overall quality of an application. Al the stakeholders, customers, managers, domain experts, developers, testers etc. are equally responsible to ensure that the product finally delivered should be secure in nature. In order to achieve this, security requirements for software serve as the first basic step for developing a secure component based system. Next step is to incorporate security attributes at architectural design level. It should be embedded in three levels, at component level which includes component certification and documentation, at level of interaction between the components, like encryption mechanisms and finally at application level like access control methods. Overall the paper focuses only on security aspect of component based application, whereas we are focused on an overall development lifecycle in this thesis.

3.4.4 Reliability of Components

Mohan and Jha in their paper [14] stress on reliability aspect of components. They propose a method to calculate reliability of the components used in a system. For their method, the have made some assumptions. First, the component based system can be decomposed easily. Second, the components do not rely on each other. Markov process is followed in flow of control between the components. In their method, they first draw a component dependency graph. Then they identify all possible paths of execution. Two paths are then selected, one path on which lies the component of whose reliability is to be found, and the other path which does not have that particular component. They find the reliability of each path and take their difference. After all paths are evaluated, average is taken to get a single value and then subtracted from one. This way, reliability of the component is found. Reliability in our case is just checked through testing. Such calculation of values is not required in our thesis.

3.4.5 Secure Sourcing of COTS Products

Most of the software applications used in organizations are bought; very few organizations build their own software. In such a scenario, Mead, Kohnke and Shoemaker are of view that such applications may contain hidden or malicious code [18]. It is not usually known that the application purchased contains components from which vendors. As a result, there is huge possibility of malicious code. To counter this, all roles are required to perform their tasks as per a given supply chain risk management (SCRM) procedure. These roles are normally customer, supplier and integrator. The paper describes ten principles according to which such a process should be governed and nine critical tasks to be performed in the given process. The proposed process includes four modules, governed by ten principles to carry out nine critical tasks. These modules are program initiation and planning (module 1), specification RFPs, contract terms (module 2), prepare response to RFP (module 3) and project/ contract management (module 4). The proposed model addresses the security concerns of applications developed through COTS components, and provide a security guideline for researchers working on development models.

3.4.6 Importance of Coordination in Agile Software Development

In a distributed environment, where different stakeholders are geographically at a distance, timely coordination is a key factor for successful projects. This study [20] analyzes agile system of working focusing on their inter dependencies and coordination required for collaboration of work. In their study, interview of eight important members was carried out. In addition, three different meetings were also monitored. After a study of two months, results were compiled by gathered data. First dependencies between distributed teams were identified. These include dependencies between vendor and co-located squad and dependencies between co-located and remote squad members. To cater for these dependencies coordination techniques were discussed. The main emphasis is on coordination with a remote COTS vendor and with a remote squad member. The focus of this paper is on all the stakeholders of a distributed environment being on same grid. It involves equal knowledge sharing and coordination between all remote stakeholders. This paper does not deal with coordination carried out at a single level to ensure harmony between all the stakeholders.

3.5 Conclusion

The literature review has been carried out keeping in view different aspects of component based development and common problems faced by developers nowadays. From the review following issues are evident:

1. Different problem areas, on which research is being carried out, have become large problem domains rather than being a sub domain of CBSE.
2. Most of the research aims at a single problem.
3. There is a lack of a framework that proposes such a development cycle that addresses development along with the two major problem domains.
4. Faster time to market is a key feature of CBSE. No focus has been put in to reduce time while proposing solutions to problems.

Chapter 4

PROPOSED CBSE FRAMEWORK – OVERVIEW

4.1 Overview

In this chapter we propose a component based development framework that will focus on the component identification and selection, and component mismatch issues. As discussed in last chapter, such a framework is lacking that addresses the major issues as well. In this chapter, while proposing various phases of our development cycle, the two issues will also be resolved in such a way that will reduce development time.

The activities involved in our proposed framework consist of two phases, ***prior to execution*** and ***after execution***. By execution we mean that system has been set into running state. Therefore, few activities will be performed before launching of project, and rest after the launch. Overview of both phases is described below:

4.2 Phase I: Prior to Execution

There are two important activities in this phase:

1. Creation of goal document
2. Creation of XML documents of each component repository

4.2.1 Goal Document

This document contains the functionalities of an overall system/ sub system. These functionalities are the outcomes of a system responding to various events. Each functionality is one goal in the goal document as shown in figure 6. In short, this document explains what behavior is expected of the system under different scenarios. This set serves as input for monitoring system behavior, also for selection of a new component in case of deviation.

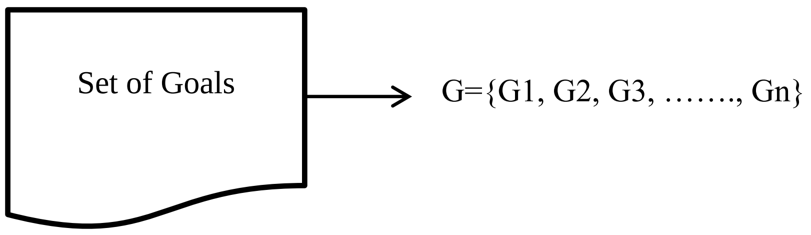


Figure 6: Goal Document

4.2.2 XML Document

Software component repositories are expressed in form of XML documents. This is a key step in our proposed framework. This step helps in two phases, searching of required component and in resolution of component mismatch issues. Therefore, it is very critical to formulate this document with accurate and precise data. As shown in figure 7, any type of repository, either local or internet, will be represented in XML format. Details will follow in subsequent chapter.

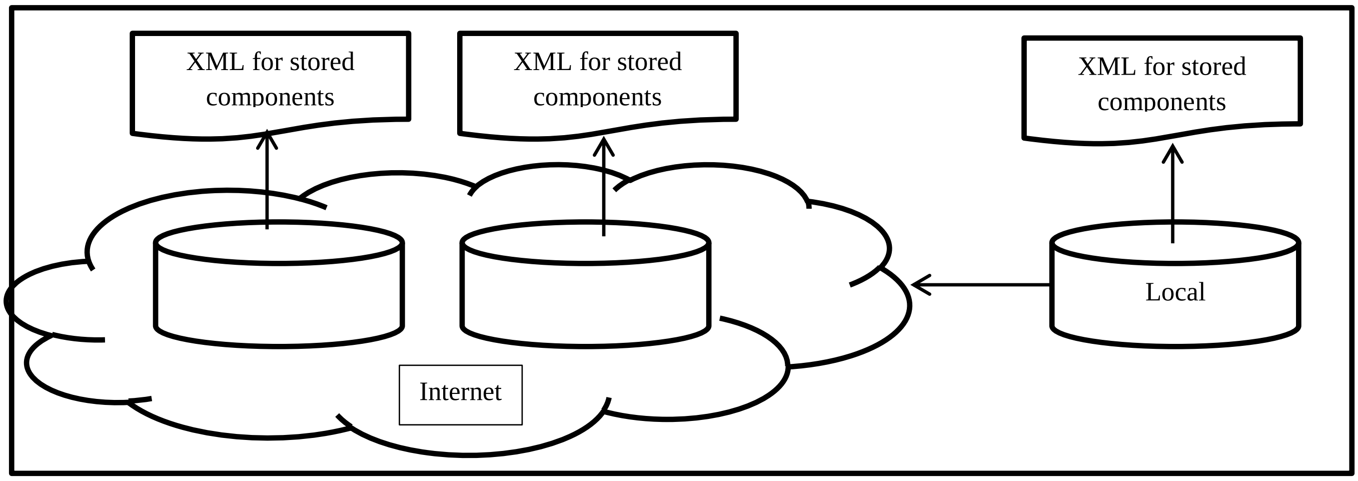

XML for stored components
XML for stored components
XML for stored components
Local
Internet

Figure 7: XML Document for Repositories

4.3 Phase II: After Execution

This second phase of our proposed framework comprises of steps that are carried out after the system is set into running state. The running system is monitored continuously for any deviation. If deviation found, event causing along with the component deviating from expected behavior are analyzed. If a need for new component of found, then search is carried out from both local and online repositories. Mismatch issues of the newly found component with existing system are removed. New component is thoroughly tested in test bed, and finally integrated into live system. Goal document is updated when required. Figure 8 shows an overview of the proposed framework.

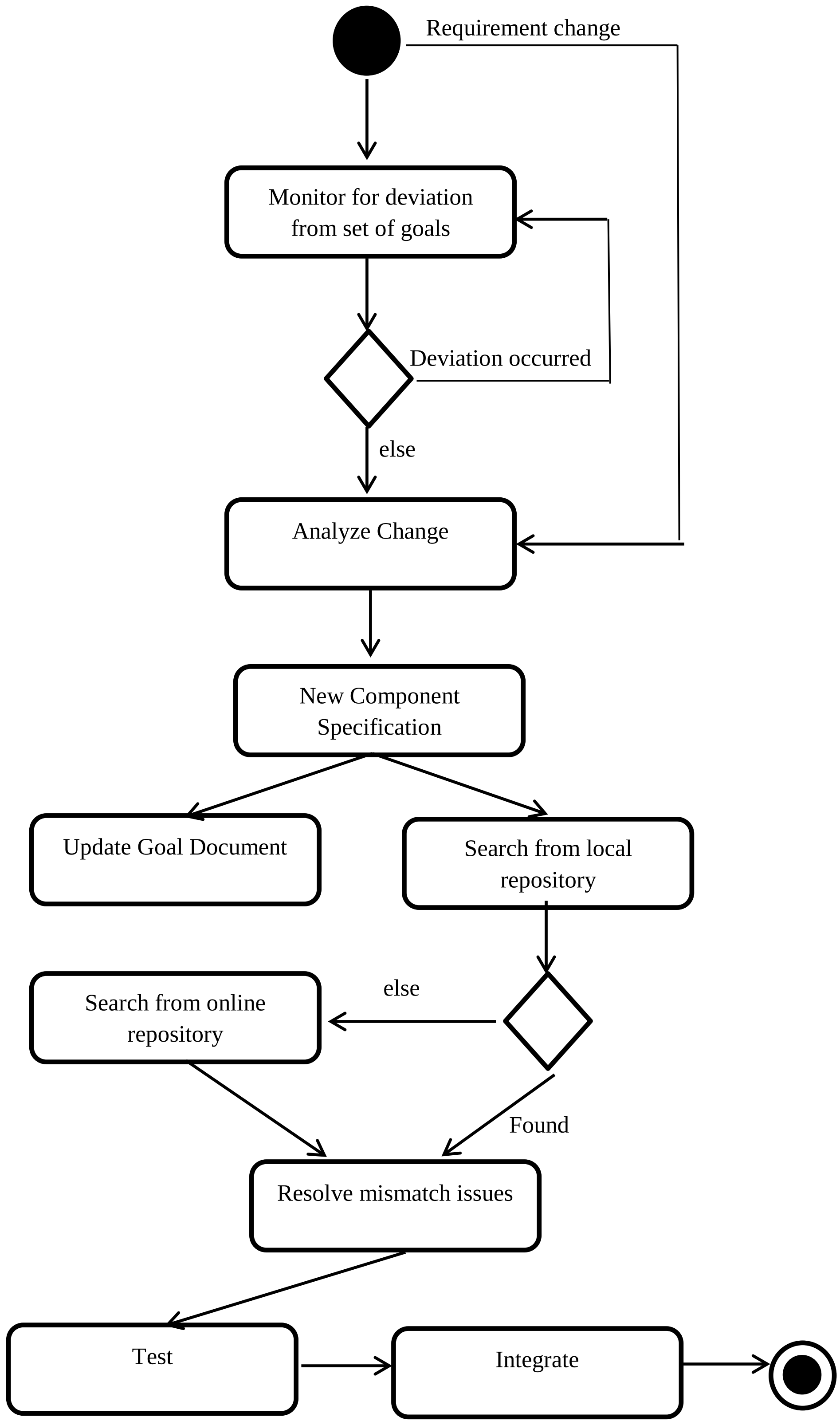

Requirement change
Monitor for deviation from set of goals
Deviation occurred
else
Analyze Change
New Component Specification
Update Goal Document
Search from local repository
Search from online repository
else
Found
Resolve mismatch issues
Test
Integrate

Figure 8: Proposed Framework

4.3.1 Set of goals

A system must have a set of goals, according to which the behavior of the running system can be monitored and analyzed. Therefore, a deliberate effort should be carried out to write these goals in a document. A system usually have multiple goals at various point of events i.e.,

$$\mathbf{G=\{G1,G2,G3,\ldots.Gn\}}$$

This set serves as input for monitoring system behavior, also for selection of a new component in case of deviation.

4.3.2 Monitor

As soon as the system is set in running state, a mechanism is set in place to monitor running system for any deviation from the defined goals. After every event ***e***, the output ***O*** of the system is cross referenced with the set of goals ***G***. The output of the system should be in accordance with the defined goals. If ***O*** is not in set ***G***, system is said to have deviated from expected behavior. The deviation is recorded and is sent for analysis phase.

4.3.3 Analysis

Analysis phase focuses on identifying component(s) that exhibited deviated behavior. The event, sub system handling the event, component whose output was unexpected, is analyzed. If a new component is required to cater the change, then specification of new component required is also evaluated. If required, Goal document is updated.

4.3.4 Search for New Component

New component is searched from local repository first. If not found locally, then search for new component is carried out from online component repository. The search is carried out using the XML documents created for each repository as explained earlier.

4.3.5 Resolution of Mismatch Issues

Semantic and data format are most common types of mismatches between components to interact with each other. The XML document created for each repository plays a vital role in this stage. We have designed the document in such a way, that each functionality

4.3.6 Testing

Once all issues resolved and the components become compatible to interact, the new component is integrated in a test environment first. The event that had caused system behavior to deviate from defined goals is recreated and the new component is tested for its functionality. Also the component is tested for its effects on overall system. To do this, new component is made to go through multiple iterations with multiple scenarios.

4.3.7 Integration

Once all possible scenarios are tested and system is found stable with the integration of new component, the new component is then introduced into the live system.

4.4 Conclusion

This chapter gives an overview of the proposed framework. All phases of the proposed framework are the same as in already followed component based development models. The difference is that it incorporates the resolution of major issues of component selection and component mismatches. Creation of XML document serves dual purpose; resolution of major issues and reduction in development time. Hence, XML document is the key entity in our proposed framework.

Chapter 5

PROPOSED CBSE FRAMEWORK – DETAILED

5.1 Overview

In this chapter, we will explain the framework proposed in the previous chapter. Our main aim is to provide a logical sequence of all processes of component based development that are involved in the lifecycle, with view to incorporate change. We do not propose a new solution to any sub problem domain. Each step is a domain itself. For example, component search and retrieval is a vast research area and already a lot of research has been carried out. Same goes for all phases. We aim to propose a framework, focusing on all phases in a way to carry out change in the application with least possible time.

5.2 Phase I: Prior to Execution

The two main activities explained in previous chapter involve two major documents. Goal document is specific to a project, whereas XML file of repositories is a file held by organization for all projects in the organization and is constantly updated.

5.2.1 Goal Document

Goal document is to be prepared for the application to be developed when functional requirements are written. Goals are written for each module of the system to be implemented. Goals can be termed as functional requirements of that module. For example, ***Save*** button has to save the information typed on form into database. This is the goal of save button.

The behavior of the particular component is to be analyzed in accordance with this goal document. Therefore, it should be created with deliberate effort. We take an example of a ***Spell Checker*** component, dependent on ***Dictionary*** component. While typing the spell checker has to correct the erroneous words. We can write the goals **G** for this component to correct the misspelled words as follows:

G= {

G1: Search Dictionary after Space character input

G2: Search Dictionary after dot character input

G3: Check for shifted characters

G4: Check for double characters

G5: Check to append a character

G6: Check to remove a character

G7: Check to switch two consecutive characters

G8: Replace the word with correct spellings

G9: Highlight the word

}

All actions performed by spell checker, should be included in this document. Any action other than defined in goal document is taken as deviation from behavior.

As per G1 and G2, the Spell checker will check each word from Dictionary component after either space or full stop is entered. The Spell checker will correct the spellings as per G3 to G7 by shifting, doubling, appending, removing or switching of characters. If word is corrected, the correct word will replace the erroneous word, otherwise will be highlighted.

5.2.2 XML file for Repositories

Large software component repositories are available online for ease of developers. Much work and research has been done for searching through these large repositories. Practically, an organization requires a certain set of components that are relevant for development due to following main reasons:

1. Every organization has an area of expertise in which they develop and deliver software products. Therefore, components required by an organization are relevant to their area of expertise; it can be web based applications, e-commerce sites, GIS, wireless communications, networking etc.
2. Every organization uses certain technologies for developing products and have relevant programmers, domain experts etc. it can be Java, .NET, python. Hence, the type of components can be categorized based on technology as well.

Based on above facts, we propose that a ***ready reckoner*** should be available with organization for easy and quick retrieval of components when required, especially for updating of a running application. It is not advisable to search through huge repositories again and again whenever a component is required. Updating of this ready reckoner is a parallel process and must be ensured by senior management that updated ready reckoner is available to developer whenever required.

In the proposed model, component repositories are expressed in XML documents which serve as ready reckoner. Generation of XML document is the backbone of the complete process. These documents are designed with a view to minimize the effort in two major stages, searching and resolution of mismatches.

Component repositories are expressed in form of XML file. Each component is written in form of XML, thus creating an entire repository in XML For example:

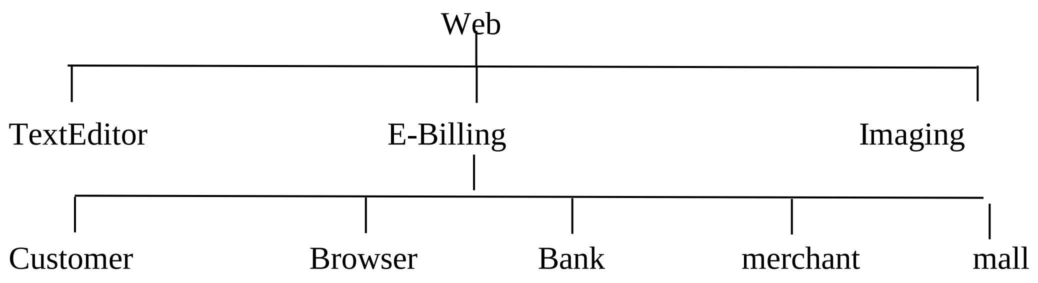
```
<Component>
    <domain>web</domain>
    <subdomain>ebilling</subdomain>
    <name>E-Billing</name>
    <main_function>payment< main_function >
    <main_function>billing< main_function >
    <main_function>accounting< main_function >
    <input>CardNumber</input>
    <input>PIN</input>
    <input>TotalAmount</input>
    <output>Result</output>
</Component>
```

XML provides a hierarchal structure, with parent child relation in which information is stored. In such a way components become easy to search and identify. Suppose we are developing an ebilling website and relevant components are required. We look up in our XML file and find the following :

```
<web>
    < TextEditor >< /TextEditor >
    < E-Billing>< / E-Billing >
            < Customer></ Customer>
            < Browser></ Browser>
            < Bank></ Bank>
            < merchant></ merchant>
            < mall ></ mall >
    < Imaging>< / Imaging>
```

</web>

This yields a tree with all components arranged in a parent childlike manner.



The tree shows that we can get all web components with sub category e-billing. Hence, expressing repositories in XML format comes very handy. It is a light weight solution, with all information organized. This makes search and retrieval of required components very quick and easy.

5.3 Phase II: After Execution

This is the main execution phase. In phase I, we did preparatory work, which is key input to the execution phase. We will go through each step as explained in previous chapter.

5.3.1 Monitor

Let us consider the example of Spell checker that we assumed earlier for creation of goal document. Assume the system is in execution state. The system is being monitored and output of each action ***e*** of the system is checked for conformity with the set of goals ***G***. pseudo code can be listed as follows:

```
FOR each action e
    IF e is in G THEN
            PRINT output
    ELSE
            THROW exception
    END IF
END FOR
```

The Spell Checker is working fine, correcting all erroneous words or highlighting them. Suddenly Spell checker stops correcting the misspelled words. It neither corrects nor highlights the misspelled words. Exception is thrown. Deviation from behavior is recorded and the process enters the Analysis phase.

5.3.2 Analysis

Thrown exceptions are logged and analyzed by developers. If the exception is a deviation:

1. Inputs are tested as per IDL specification provided by vendor, that in no case invalid input is being generated and fed into the component.
2. If inputs are valid and it is found that the component is not giving the desired output, XML schemas are then searched for new component.
3. If the exception is a required change, following is analyzed:
    a. Nature of change
    b. Impact on system
    c. Amount of effort to be put in

If it is approved to make the change, new specification of component is written. XML schemas are searched for required component.

Continuing our Spell checker example, on observing the behavior and exceptions, it is found that Spell Checker is dependent on another component Dictionary as shown in figure 9. Spell checker is unable to access the Dictionary component. On further debugging, when Dictionary component was accessed and analyzed, it was found that Dictionary has been updated and there is a version mismatch.

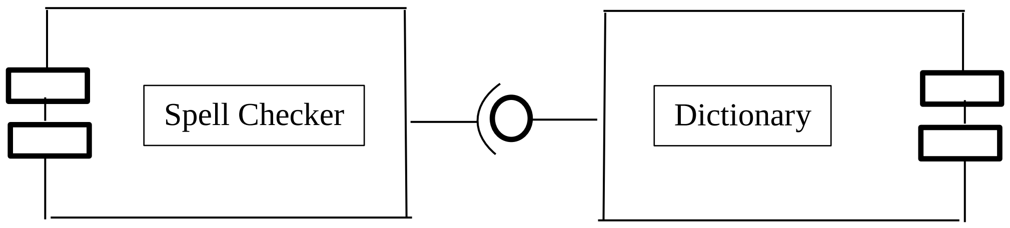


Figure 9: Spell Checker and Dictionary Components

Hence, the updated version of current Dictionary component is required **OR** new component is needed to be found.

5.3.3 Search for New Component

XML file is checked for new component. First local repository is searched for required component. If not found, then online repositories are searched through their XML files. This is one of the two important phases where a XML document of repository, created with extreme deliberate effort saves tremendous time and effort. Updating of XML file is a parallel process and not assumed a part of this framework. It is required to be carried out frequently, as explained in section 5.2.2. The amount of effort required should be spent on creation of XML document; hence time and effort are saved during execution phase.

In our current example of Spell checker, either updated version of current Dictionary component is acquired **OR** new Dictionary component acquired. Suppose we have following two components of dictionaries available in our local repository. One is ***Oxford Dictionary*** and second is ***Webster Dictionary,***

```
< dictionary>
    <domain>grammer</domain>
    <subdomain>texteditor</subdomain>
    <name>oxfordDictionary</name>
    <version>2.3</version>
    <main_function>meaning< main_function >
    <main_function>spellcheck< main_function >
    <main_function>synonym< main_function >
    <input>word, function </input>
    <output>word, function </output>
</ dictionary>

< dictionary>
    <domain>grammer</domain>
    <subdomain>texteditor</subdomain>
    <name>websterDictionary</name>
    <version>4,1</version>
    <main_function>meaning< main_function >
    <main_function>synonym< main_function >
    <input>word, function </input>
    <output>word, function </output>
</ dictionary>
```

XML shows name, version, functionality, inputs and outputs of a component. Above XML shows that Webster Dictionary component does not support the spell check functionality. It only returns meaning and synonym of a given word, whereas, for each word, the Oxford dictionary provides closest resembling word. Thus, we will select Oxford Dictionary for our Spell checker component.

5.3.4 Resolution of Mismatch Issues

Mismatches can be of semantic or behavioral nature. In current example of Spell checker, the change of version has resulted in behavioral mismatch. Again, a well formulated XML file can help save a lot of time. Consider the following dictionary components again:
Suppose we have following two components of dictionaries available in our local repository. One is ***Oxford Dictionary*** and second is ***Webster Dictionary,***

```
< dictionary>
    <domain>grammer</domain>
    <subdomain>texteditor</subdomain>
    <name>oxfordDictionary</name>
    <version>2.3</version>
    <main_function>meaning< main_function >
    <main_function>spellcheck< main_function >
    <main_function>synonym< main_function >
    <input>word, function </input>
    <output>word, function </output>
</ dictionary>
```

```
< dictionary>
    <domain>grammer</domain>
    <subdomain>texteditor</subdomain>
    <name>websterDictionary</name>
    <version>4,1</version>
    <main_function>meaning< main_function >
    <main_function>synonym< main_function >
    <input>word word, function</input>
    <output>word, function </output>
</ dictionary>
```

XML shows that Webster dictionary takes two words as input, whereas our system only provides a single word along with function name and expects a word as a result. We can either adapt our code to provide two word inputs, or simply go for Oxford dictionary. Hence, many of mismatch issues are evident from XML file, and one can easily select a component by knowing its pros and cons.

5.3.5 Testing

Component testing is performed according to standard component testing techniques. Each component is to be tested separately (component testing in small CTIS).

5.3.6 Integration

When component is tested separately, it is then integrated into the system. Component testing in large (CTIL) is then carried out, to ensure its interaction with rest of the components, and its effect on the entire application as a whole.

5.4 Application of Six Sigma

Six Sigma is a structured approach that aims at improving processes by reducing defects in system being developed and cost. Hence, it increases customer satisfaction. Our proposed work is also based on reducing cost and development time, thus making the developed product readily available to customer. Hence, we can relate phases of our proposed framework with five step process of Six Sigma as shown in figure 10 below:

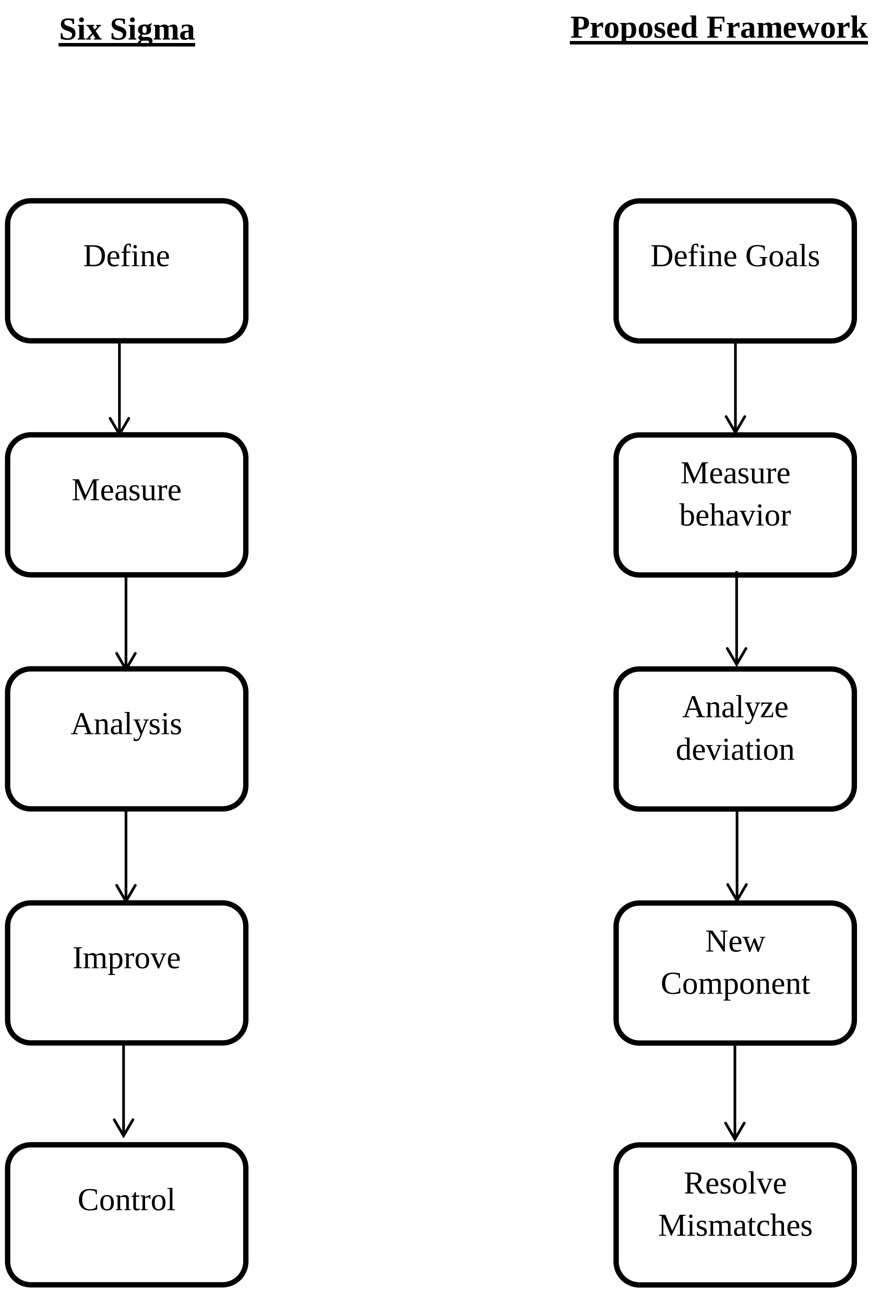


Figure 10: Proposed Framework with Six Sigma

5.5 Conclusion

The proposed framework has been described in detail in this chapter. The most important is the creation of XML document that has been described in detail. Rest phases are somewhat similar to conventional software development. Time is being consumed when faced with challenges like finding a suitable component for the project, then making it able to interact with rest of the components in the system. XML document resolves this issue, by offering a related set of components, written in a manner that spells out its functionality, inputs and outputs described in component interface. Hence, the desired aim of addressing issues with minimum possible time is achieved.

Chapter 6

PROPOSED CBSE FRAMEWORK – EVALUATION

6.1 Overview

In this chapter we will apply our proposed framework to a running component based system. We will try to prove that by following the proposed model, search time for a required component and making it compatible can be lessened; a change can be incorporated into a running component based system aiming at reduction of time consumption and lessening the chances of errors if steps of the framework are followed.

6.2 Case Study

We take a hypothetical example of a **Tour Management System** as a test case. A change is required in a running tour management system and needs to be incorporated. Let us first consider what all components are involved in a successful running system. In this system we have following components:

1. Tour information system
2. Hotel booking
3. Travel agency
   a. Car
   b. Air
4. GIS
5. Payment component
6. Charges component

These components work together, as shown in figure 10, in the following way to perform the desired functionality:

1. Tour information offers various packages, giving details of places to visit, their addresses, maps for guidance and cost. A tourist books a suitable package for himself.
2. After selecting package, he is directed towards hotel reservation component. All available hotels with their facilities and rates are offered. The location of hotels can be viewed through maps. The tourist makes a suitable reservation for desired number of days. Tourist can skip this booking if he has own arrangement.
3. The third step is booking of transport, either a car or air tickets. The tourist selects a suitable car/ air package. Tourist can skip this booking if he has own arrangement.

4. After making necessary bookings and facilities, charges component presents final bill.
5. To proceed with payment, payment component processes credit card credentials of tourist. Payment is done and process is complete.
6. All the maps shown throughout the system are through the GIS component.

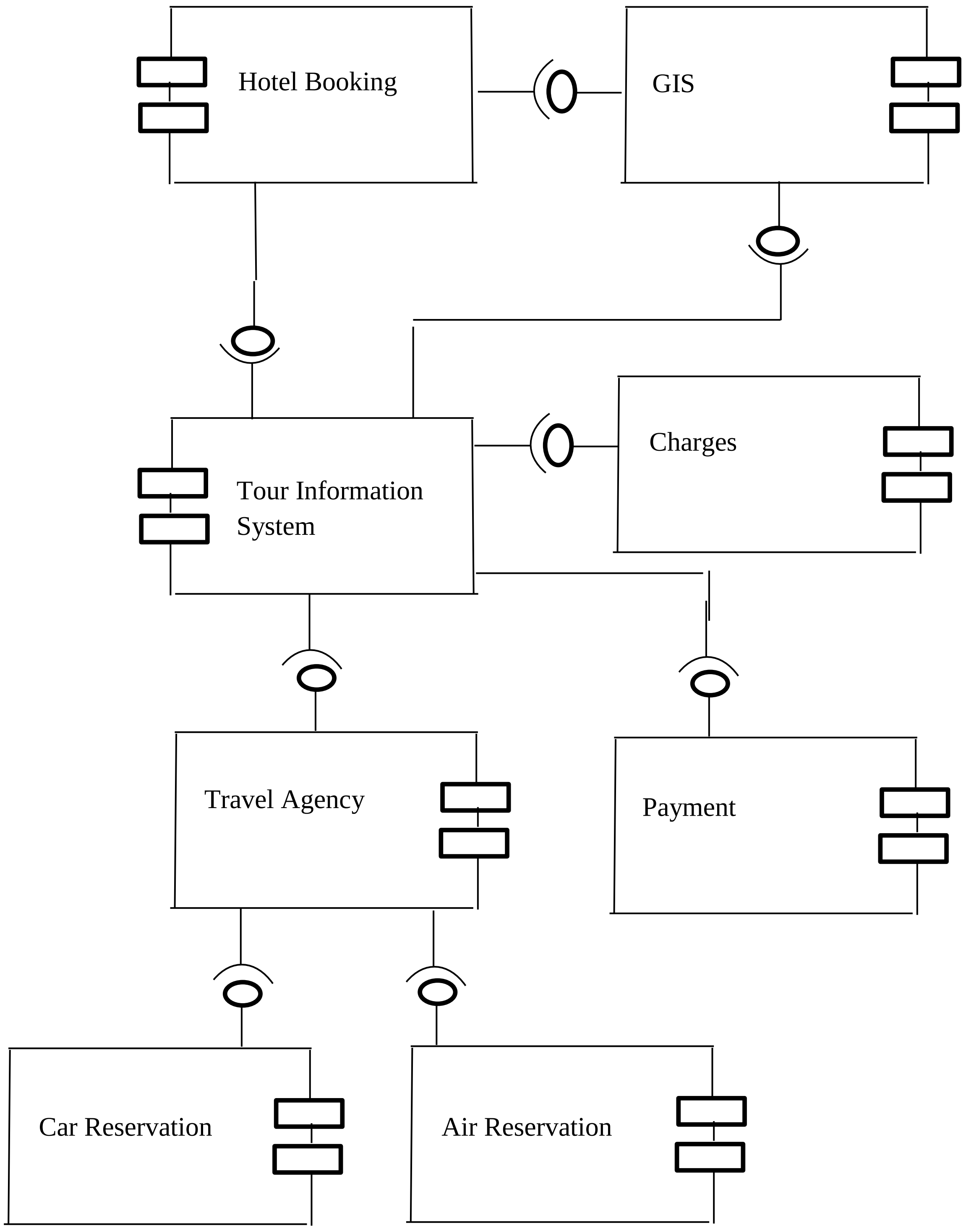


Figure 11: Tour Management System

6.3 Requirement Change

The GIS component of the Tour management system is designed to give an attractive yet complete picture of the tourism place to the tourist so that he can select a suitable package for himself. The present component performs the following functionality:

1. Search for a location
2. Nearby restaurants, gas stations, attractions
3. Directions to reach a destination
4. Estimated time required
5. Shortest path to destination
6. Fastest path to destination

As far as booking is concerned, the GIS component serves well. But when on ground, a tourist usually buys maps of the tourism place. Our aim is to provide such a map that will help the tourist to explore the markets, gardens museums etc. by themselves. If exploring on foot, the tourist should be able to select a route passing through most of the places of interest. So they require a complete guide with proper roads shown as shown in figure 11.

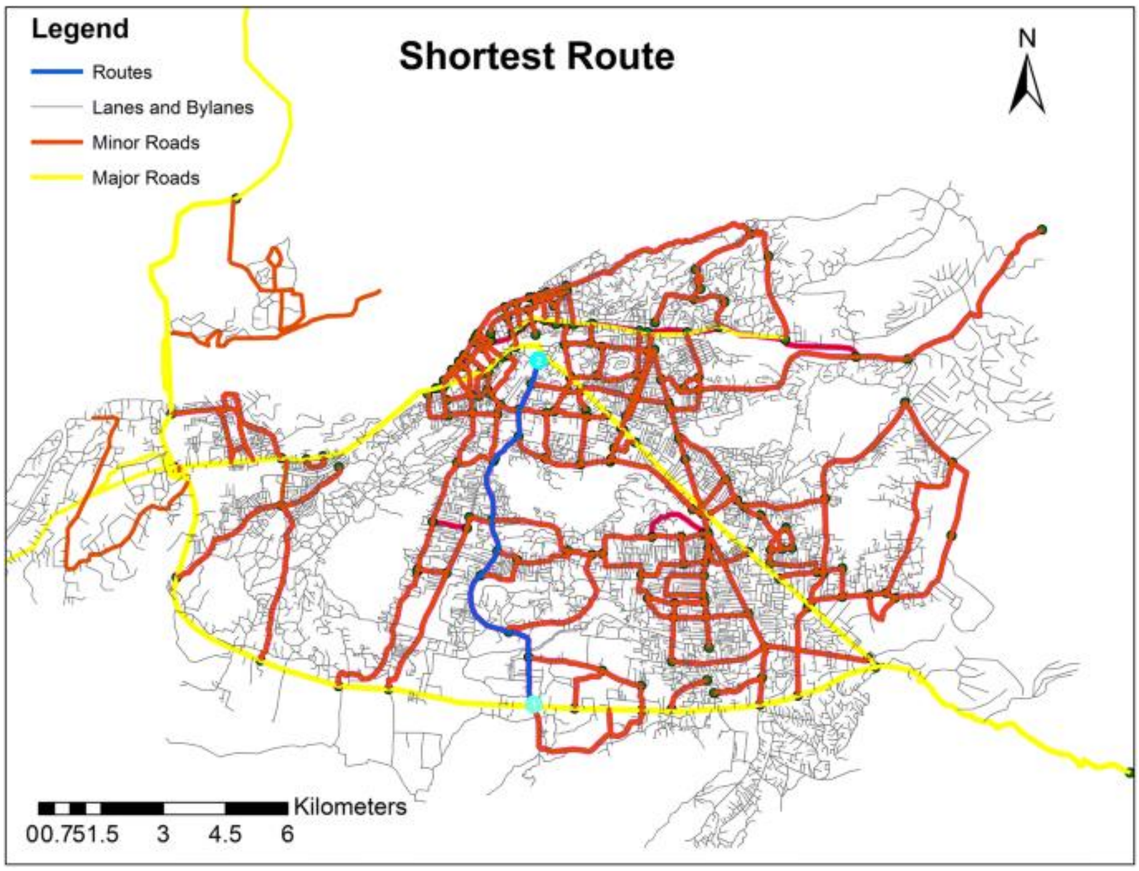


Figure 12: Road Network View

6.4 Requirement Handling Through Proposed Framework

We will apply the proposed model step by step to change the present GIS component with the one that provides **road network view** as well.

6.5 Phase I: Prior to Execution

The two steps involved in this phase will produce documents as the system was conceived. This phase is not meant for requirement change. The documents produced in this phase will be updated whenever a change is required in phase II.

6.5.1 Goal Document for GIS Component

We will consider the set of goals defined for the GIS component. As described earlier, the main functionalities required by a system/ sub system are defined as goals of that system. In this case, our required set of goals is as follows:

G = {

G1:	Search for a location
G2:	Nearby restaurants, gas stations, attractions
G3:	Directions to reach a destination
G4:	Estimated time required
G5:	Shortest path to destination
G6:	Fastest path to destination
G7:	Display map

}

6.5.2 XML Document

XML document for each repository contains list of GIS components with functionality that comes under the business functions of a company. Hypothetically, we have created two xml files for repositories **Maven** (online repository for Java Maven components) and **NuGet** (online repository for .NET components).

Note: The repositories and component names displayed below are from real time online repositories. However for understanding purposes, other parameters have been changed and assumed.

Each xml file currently holds one component for GIS. Both files are displayed below:

Maven Repository

```
<mavenRepository>
        <componentRepository category="GIS">
                <component >
                        <name>gisroad</name>
                        <version>17.0</ version>
                        <dependency>org.arakhne.afc</dependency>
                        <description>Base elements for creating data structures that
                        represent a road network</description>
                        <functionality>permits to display a road network</functionality>
                        <functionality>describes a path inside a road
                        network</functionality>
                        <functionality>represents the connection point inside a road
                        network</functionality>
                        <input>int startpoint</input>
                        <input>int endpoint</input>
                        <output>image roadmap</output>
                </component>
        </componentRepository>
</ mavenRepository >
```

NuGet Repository

```
<nugetRepository>
        <componentRepository category="GIS">
                <component >
                        <name>roadMap </name>
                        <version>100.14.1</ version>
                        <dependency> Esri.ArcGISRuntime </dependency>
                        <description> geospatial and location intelligence </description>
                        <functionality> MapView control used for the display of map
                        layers and information in 2D </functionality>
                        <functionality> SceneView used for the display of 2D layers and
                        3D scene layers and information in 3D </functionality>
                        <functionality> Platform-specific image support</functionality>
                        <input>int startpoint</input>
                        <input>int endpoint</input>
                        <output>image roadmap</output>
                </component>
        </componentRepository>
</ nugetRepository >
```

6.6 Phase II: After Execution

The system is in running state.

6.6.1 Monitor

In our test case, change is due to user feedback, not from system malfunctioning. With the current running system, the tourists are dependent on tourism maps/ guides that they have to purchase separately. This scenario does not go well with a management system. The company receives negative feedback from customers that forces them to upgrade their GIS component.

6.6.2 Analysis

Analysis of the change required reveals that a feature, **Download Guide** should be added. The resulting map should be in road network showing all roads in the area. All important places including markets, restaurants, gas stations, museums, libraries, parks etc should be properly displayed. A tourist should get a clear picture that if on foot, what route will be best for his site seeing. When change was analyzed, it urged a new component

The change is added in set of goals. Updated set of goals become:

G = {

G1:	Search for a location
G2:	Nearby restaurants, gas stations, attractions
G3:	Directions to reach a destination
G4:	Estimated time required
G5:	Shortest path to destination
G6:	Fastest path to destination
G7:	Display map
G8:	Download map in road network view

}

6.6.3 Search for New Component

The XML documents are searched for components. All repositories are searched for components under category GIS. Search from Maven repository:

Select * from mavenRepository where category= GIS

Result Table

Name	Version	Description	Dependency	Functionality
Gisroad	17	Base elements for creating data structures that represent a road network	org.arakhne.afc	permits to display a road network describes a path inside a road network represents the connection point inside a road network

Table 2: Component Search for GIS Components

Similarly, search from NuGet repository:

Select * from nugetRepository where category= GIS

Result Table

Name	Version	Description	Dependency	Functionality
roadMap	100.14.1	geospatial and location intelligence	Esri.ArcGISRuntime	MapView control used for the display of map layers and information in 2D SceneView used for the display of 2D layers and 3D scene layers and information in 3D Platform-specific image support

Table 3: Component Search for GIS Components

From the two searches, the component **gisroad** from Maven repository is selected as it best fulfills the requirement.

6.6.4 Resolution of Mismatch Issues

In this phase also, the xml document will decrease our time tremendously. The specification of the IDLs is well laid out in the form of inputs and outputs, which makes the process of selection of component much faster. Out of multiple components, the developer exactly knows which component will give the best suited results.
As the inputs and outputs are well documented, and functionality is clearly written,
The semantic issues are overcome easily by description and functionality. The syntax issues are resolved by inputs and outputs.

6.6.5 Testing

Once all issues resolved and the components become compatible to interact, the new component is integrated in a test environment first. Also the component is tested for its effects on overall system. To do this, new component is made to go through multiple iterations with multiple scenarios. During this phase, the new following functionalities are tested:

1. The goal document is verified to be updated and includes the new requirement change:

   G8: Download map in road network view
2. The new GIS component is tested against the new goal. Whenever **Download Guide** button is pressed, the map is saved in **road network view**.
3. In the downloaded map, all the roads of the area are highlighted.
4. All important places are shown/ marked on each road.
5. Map is tested for saturation of information. Names of places/roads should not be cluttered.
6. Other maps in application are tested for to being displayed in normal view. Road network view is only meant for the guide map.

6.6.6 Integration

Once all possible scenarios are tested and system is found stable with the integration of new component, the new component is then introduced into the live system.

6.7 Conclusion

The proposed framework has been evaluated in this chapter by considering a case study of Tour Management System. A dry run down of the complete framework was carried out. The desired results are evident from the use of XML document. It was easy to locate all candidate components from the repositories and then finally selecting the best suited component. Moreover, it was easy to make the GIS component interactive as its inputs and outputs were clearly stated in the XML document. Hence, we can say, that the proposed framework offers a complete development lifecycle, focusing on two major issues, and resolving hem in a way that is not time consuming and does not delay the project.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Overview

In this chapter, we conclude the topic the thesis concludes with a summary of all the work done in this research. It provides the conclusion of the proposed model. Also, list the possible future work to explore.

7.2 Conclusion

In this thesis, we have proposed a framework to incorporate a change in running component based software application to achieve following:

1. A ready reckoner of components should be available for developers/ domain experts at any time they require, either start of project or to carry out a change during execution phase.
2. The ready reckoner, XML document in our proposed framework, represents set of components relevant to an organization based on their technology expertise. Hence, it saves tremendous time while searching for a component of certain specification.
3. The XML document is designed with a view to represent meta-data of components in such a way so that it helps not only in search phase, but its information is enough to resolve mismatch issues.

The proposed framework is in line with Six Sigma methodology in view to curtail defect rates, thus promising to carry out a change in a systematic manner giving desired results with fewer chances of errors.

7.3 Limitations

In proposed framework, the component is selected based on technical requirements only. ***Cost*** and ***quality attributes*** have not been dealt with in our proposed framework.

7.3.1 Cost Issues

In proposed work, while selecting components, no focus has been paid on cost of components. Cost plays a vital role in component selection. Chatzipetrou et al investigated that while going for a component selection, what will the criteria of utmost importance for domain specialists [6]. To obtain evidence, they conducted a survey involving specialists of different domains. Not so surprising, they found that it was the cost of the components that was a key factor in decision making of practitioners.

7.3.2 Non Functional Requirements

In normal practice, functional requirements are the driving factor in identification and selection of components. Those components are selected, for which, minimal code is required to be written to make the component compatible with rest of the components of the system. Due to this practice, non-functional requirements are often neglected by the developers. In the same manner, the proposed framework has not addressed the quality attributes of the components, affecting performance of the overall system and has focused only on functional requirements.

7.4 Future Work

The framework proposed can be extended by overcoming the above mentioned limitations.

7.4.1 Cost Evaluation

Basic cost estimation techniques [23] can be applied on the proposed model to evaluate cost of component:

1. Algorithmic cost modeling
2. Expert judgment
3. Estimation by analogy
4. Parkinson's Law
5. Pricing to win

7.4.2 Quality Attributes

Software frameworks are supposed to conform to ISO IEC 25010 or ISO IEC 9126. The framework needs to be tested for conformity with these ISO standards. Nonconformance, if found, should be removed by modifying the framework wherever applicable. This will strengthen the framework and add credibility.

REFERENCES

1. AGORA Search Engine Documentation. https://authecesofteng.github.io/agora/, (accessed August 30, 2022).
2. Yao, Lawrence & Rabhi, Fethi & Peat, Maurice. (2014). Supporting Data-Intensive Analysis Processes: A Review of Enabling Technologies and Trends. 10.4018/978-1-4666-6178-3.ch019.
3. Luqi, & Zhang, L. (2007). Software Component Repositories. Wiley Encyclopedia of Computer Science and Engineering. doi:10.1002/9780470050118.ecse378
4. S. Tang and Q. Liu, "Supporting Integration of COTS Components from a Perspective of Self-Adaptive Software Architecture," 2013 IEEE 37th Annual Computer Software and Applications Conference, 2013, pp. 706-713, doi: 10.1109/COMPSAC.2013.112.
5. Umran Alrubaee, A., Cetinkaya, D., Liebchen, G., & Dogan, H. (2020). A Process Model for Component-Based Model-Driven Software Development. Information, 11(6), 302.
6. Rosa, L., Rodrigues, L., Lopes, A., Hiltunen, M., & Schlichting, R. (2012). Self-management of adaptable component-based applications. IEEE Transactions on Software Engineering, 39(3), 403-421.
7. Abraham, B. Z., & Aguilar, J. C. (2007, May). Software component selection algorithm using intelligent agents. In KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications (pp. 82-91). Springer, Berlin, Heidelberg.
8. Gholamshahi, S., & Hasheminejad, S. M. H. (2019). Software component identification and selection: A research review. Software: Practice and Experience, 49(1), 40-69.
9. Shukla, R., & Marwala, T. (2013). Component Based Software Development Using Component Oriented Programming. In Proceedings of International Conference on Advances in Computing (pp. 1125-1133). Springer, New Delhi.
10. Capilla, R., Gallina, B., Cetina, C., & Favaro, J. (2019). Opportunities for software reuse in an uncertain world: From past to emerging trends. Journal of software: Evolution and process, 31(8), e2217.
11. T. Lu, C. Liu, H. Duan and Q. Zeng (2020). "Mining Component-Based Software Behavioral Models Using Dynamic Analysis," in IEEE Access, vol. 8, pp. 68883-68894, doi: 10.1109/ACCESS.2020.2987108.

12. A. Segal-Raviv, I. Hadar and M. Levy, "Facilitating Collaboration between COTS Stakeholders via Principles of Advanced ISD Methods: The Vendor Perspective," 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, 2015, pp. 29-35, doi:10.1109/CHASE.2015.11.
13. Jha, S.K., Mishra, R.K. (2019). Predicting and Accessing Security Features into Component-Based Software Development: A Critical Survey. In: Hoda, M., Chauhan, N., Quadri, S., Srivastava, P. (eds) Software Engineering. Advances in Intelligent Systems and Computing, vol 731. Springer, Singapore. https://doi.org/10.1007/978-981-10-8848-3_28
14. A. Mohan and S. K. Jha, "Predicting and Accessing Reliability of Components in Component Based Software Development," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), 2019, pp. 1110-1114, doi: 10.1109/ICCS45141.2019.9065290.
15. D. Negi, Y. S. Chauhan, P. Dimri and A. Harbola, "An Analytical Study of Component-Based Life Cycle Models: A Survey," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), 2015, pp. 746-750, doi: 10.1109/CICN.2015.152.
16. H. Sastypratiwi and Y. Yulianti, "Web Application Development using MVC-component-based approach," 2019 International Conference on Data and Software Engineering (ICoDSE), 2019, pp. 1-5, doi: 10.1109/ICoDSE48700.2019.9092609.
17. J. P. Carvallo, X. Franch and C. Quer, "Determining Criteria for Selecting Software Components: Lessons Learned," in IEEE Software, vol. 24, no. 3, pp. 84-94, May-June 2007, doi: 10.1109/MS.2007.70.
18. N. R. Mead, A. Kohnke and D. Shoemaker, "Secure Sourcing of COTS Products: A Critical Missing Element in Software Engineering Education," 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), 2020, pp. 1-5, doi: 10.1109/CSEET49119.2020.9206233.
19. Selić, B. (2021). Specifying dynamic software system architectures. Software and Systems Modeling, 20(3), 595–605. doi:10.1007/s10270-021-00875-0
20. Buchan, Jim; Talukder, A.B.M. Nurul Afser; and Senapathi, Mali, "Coordination in Distributed Agile Software Development: Insights from a COTS-based Case Study" (2019). ACIS 2019 Proceedings. 100.

21. Bali, V., Bali, S., & Madan, S. (2018). IFSOM: A Two-Phase Framework for COTS Evaluation and Selection. Lecture Notes in Networks and Systems, 871–888. doi:10.1007/978-981-13-1217-5_86
22. Anas Shatnawi, Hudhaifa Shatnawi, Mohamed Aymen Saied, Zakarea Al Shara, Houari Sahraoui, and Abdelhak Seriai (2018). Identifying software components from object-oriented APIs based on dynamic analysis. In Proceedings of the 26th Conference on Program Comprehension (ICPC '18). Association for Computing Machinery, New York, NY, USA, 189–199. https://doi.org/10.1145/3196321.3196349
23. Singal, P., Kumari, A. C., & Sharma, P. (2020). Estimation of software development effort: A Differential Evolution Approach. Procedia Computer Science, 167, 2643-2652.
24. Hoque, T., SLPSK, P., & Bhunia, S. (2020). Trust Issues in COTS: The Challenges and Emerging Solution. Proceedings of the 2020 on Great Lakes Symposium on VLSI. doi:10.1145/3386263.3407654
25. Rani, A., Mishra, D., Omerovic, A. (2022). Multi-vendor Software Ecosystem: Challenges from Company' Perspective. In: Rocha, A., Adeli, H., Dzemyda, G., Moreira, F. (eds) Information Systems and Technologies. WorldCIST 2022. Lecture Notes in Networks and Systems, vol 470. Springer, Cham. https://doi.org/10.1007/978-3-031-04829-6_34
26. Wynn, D. C., & Clarkson, P. J. (2018). Process models in design and development. Research in Engineering Design, 29(2), 161-202.
27. K. R. Sekar, J. Sethuraman and R. Manikandan, "A Novel Software Component Selection Through Statistical Models," 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018, pp. 1391-1396, doi: 10.1109/ICOEI.2018.8553696.
28. Chatzipetrou, P., Papatheocharous, E., Wnuk, K. et al. "Component attributes and their importance in decisions and component selection". Software Qual J 28, 567–593 (2020). https://doi.org/10.1007/s11219-019-09465-2
29. Mehta P., Tandon A., Sharma H. (2022) Integration of FAHP and COPRAS-G for Software Component Selection. In: Aggarwal A.G., Tandon A., Pham H. (eds) Optimization Models in Software Reliability. Springer Series in Reliability Engineering. Springer, Cham. https://doi.org/10.1007/978-3-030-78919-0_12

30. Cortellessa, Vittorio, Fabrizio Marinelli, and Pasqualina Potena. "Automated selection of software components based on cost/reliability tradeoff", in European Workshop on Software Architecture, pp. 66-81. Springer, Berlin, Heidelberg, 2006.
31. P. Salvaneschi, "Emerging Structures in Information Systems: A SOS Approach," 2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES), 2019, pp. 26-33, doi: 10.1109/SESoS/WDES.2019.00012.
32. H. -M. Chiao, W. -H. Huang and J. -L. Doong, "Innovative research on development game-based tourism information service by using Component-based Software Engineering," 2017 International Conference on Applied System Innovation (ICASI), 2017, pp. 1565-1567, doi: 10.1109/ICASI.2017.7988227.
33. S. U. Khan, A. W. Khan, F. Khan, M. A. Khan and T. K. Whangbo, "Critical Success Factors of Component-Based Software Outsourcing Development From Vendors' Perspective: A Systematic Literature Review," in IEEE Access, vol. 10, pp. 1650-1658, 2022, doi: 10.1109/ACCESS.2021.3138775.
34. Rana, T. EX-MAN Component Model for Component-Based Software Construction. Arab J Sci Eng 45, 2915–2928 (2020). https://doi.org/10.1007/s13369-019-04213-x
35. Belli, F., Quella, F. (2021). Software Reuse Technologies. In: A Holistic View of Software and Hardware Reuse. Studies in Systems, Decision and Control, vol 315. Springer, Cham. https://doi.org/10.1007/978-3-030-72261-6_4
36. Banerjee, P., & Sarkar, A. (2018). Quality evaluation of component-based software: an empirical approach. International Journal of Intelligent Systems and Applications, 11(12), 80.
37. Ali, S., Hafeez, Y., Humayun, M. et al. Towards aspect based requirements mining for trace retrieval of component-based software management process in globally distributed environment. Inf Technol Manag 23, 151–165 (2022). https://doi.org/10.1007/s10799-021-00343-7
38. Lau, K. K., & Cola, S. D. (2018). AN INTRODUCTION TO COMPONENT-BASED SOFTWARE DEVELOPEMENT.
39. Tiwari, U.K., Kumar, S. & Matta, P. Execution-history based reliability estimation for component-based software: considering reusability-ratio and interaction-ratio. Int J Syst Assur Eng Manag 11, 1003–1019 (2020). https://doi.org/10.1007/s13198-020-01035-1

40. Diwaker, C., Tomar, P., Solanki, A., Nayyar, A., Jhanjhi, N., Abdullah, A., & Supramnian, M. (2019). A New Model for Predicting Component-Based Software Reliability using Soft Computing. IEEE Access, 1–1. doi:10.1109/access.2019.2946862
41. Jha, S. K., & Mishra, R. K. (2016). Multi criteria based retrieval techniques for reusable software components from component repository. International Journal of Engineering Applied Sciences and Technology, 6(1), 88-91.
42. Zhuge, H. (2000). A problem-oriented and rule-based component repository. Journal of Systems and Software, 50(3), 201–208. doi:10.1016/s0164-1212(99)00097-7
43. Nassiri, H., Machkour, M., & Hachimi, M. (2018). One query to retrieve XML and Relational Data. Procedia Computer Science, 134, 340-345.