

SOFTWARE REUSABILITY: A FRAMEWORK USING SOFTWARE COMPONENTS AND REUSABLE ASSETS

¹A.RAVI, ²DR.K.NIRMALA

¹Research Scholar Ms university Tirunelveli.

²Assoc. Prof., Department of Computer Science Quaid-E-Millath Government College for Women(Autonomous) Chennai-600002

E-mail: ¹gangai_ravi@yahoo.com, ²nimimca@yahoo.com

ABSTRACT

As everything becomes computerized, more and more software have to be evolved. As the software grows rapidly, many problems have to be emerged in terms of software development and maintenance. So the Software developers have to learn more and new information in terms of their process in different areas. This information has to be managed for long time to make it use in various software. This kind of using the same information for multiple processes is termed as Reuse. In Software Engineering, this concept of reuse has been implemented very much to reduce the implementation cost and for efficient maintenance. Reusability is termed as a part or a segment of source code can be used again in order to develop new functionalities with slight or no modification. Software Reusability is the process of developing new software from existing software with slight modification in the existing one in order to adopt it for new one. In our paper, we have to implement this software reusability concept with efficient methodology and algorithm. We have to propose a methodology to develop a software component that act as a Reuse Asset which can be used to develop various software with less cost.

Keywords: *Reusability, Reuse Asset, Software Component, Software Development, Software Engineering, Software Reusability.*

1. INTRODUCTION

1.1 Software Engineering

Software Engineering is the study and application of engineering to the design, development and maintenance of software. The formal definitions of Software Engineering are:

- “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”
- “an engineering discipline that is concerned with all aspects of software production”
- “the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines”

A person who develops software is known as Software Developer or Software Engineer. In other

words, a Software Engineer creates software using methods that make it better quality. Better quality software is easier to use and the code is easier to understand, to maintain, and to add new features.

1.1.1 Phases Involved in Software Creation

Software Engineering can be a difficult work that needs a set of software engineer to develop software. Developing software is not at all an easy task. It consists of number of phases such as:

- Analysis / Requirements
- Design
- Implementation
- Testing / Verification
- Maintenance



Figure 1: Different Phases in Software Development

- **Analysis / Requirements** say what the software should do. It can be taken by discussing with the client.
- **Design** says what the different parts of the software are, and how they talk together.
- **Implementation** is to write the code for each component of the software. Code is what tells the computer exactly what to do at each step.
- **Testing** is done to see if the components meet the requirements and that the system as a whole meets the requirements.
- **Maintenance** is the part or all of this process can repeat if bugs are found or new requirements are needed.

All these phases perform various activities in developing the software and so all phases have equal importance. Of those phases, the Maintenance phase becomes a tedious one, since it maintain the software lifelong by managing it based on the customer requirements.

1.2 Reusability

In Software Engineering, reusability is the likelihood that a segment of source code can be used again to add new functionalities with slight or no modification. This kind of reusability is implemented in the programming languages such as CPP and Java, which is named as **Subroutines** or **Functions** or **Methods**. A chunk of code which is used often is organized under these modules as layers. When that code needs to be used in the program, it can be called upon. Such kind of reusability is termed as **Code Reusability**. The purpose of reusability implies some explicit management of build, packaging, distribution, installation, configuration, deployment, and

maintenance and upgrade issue. If these issues are not considered, software may appear to be reusable from design point of view, but will not be reused in practice. When this reusability is not considered, then the developing software grows larger in size and appears to be a difficult one.

This will be considered in our research work and we have to propose a methodology to create a software component in order to develop much software from the existing software using various methodology. That is, by using our methodology, if a software engineer develops software, it can be maintained in an efficient way to develop new software from it and it is known as **Software Reusability**. These are all discussed in our proposed methodology with suitable algorithm.

1.3 Software Reusability

Software Reusability is generally considered a way to solve the software development crisis. When we solve a problem, we try to apply the solution to similar problems because that makes our task simple and easy. This Software reusability can improve software productivity. Software reuse has become a topic of much interest in the software community due to its potential benefits, which include increased product quality and reduced product cost and schedule. The most substantial benefits derive from a product line approach, where a common set of reusable software assets act as a base for subsequent similar products in a given functional domain. The upfront investments required for software reuse are considerable, and need to be duly considered prior to attempting a software reuse initiative.

Software reuse is the process of implementing or updating software systems using existing software components. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge.

By considering all these things, we propose a methodology to reuse the software in a much better way. The algorithm in this paper, performs well to demonstrate how the software can be reusable and

the performance evaluation also be given in the experimental results section. Our paper will be developed by analyzing some of the relevant papers regarding this software reusability. These are all discussed in the following sections.

The need of this research work is to develop an enhanced project in order to reuse the software which yields benefits such as reduced cost, less time and minimum man power. This paper fills the gap of previous research work by providing more advantages over number of employee and the time duration in a very exact manner. Our research work is being developed under the consideration of each and every phases of the software life cycle. By enhancing these phases, we implement our proposed algorithm to reuse the software. The aim of the paper can be achieved by implementing two important techniques in this paper such as: *Impact Analysis Document* and *RTM (Requirement Traceability Matrix)*.

2. LITERATURE SURVEY

Suresh et al, in paper [1] stated that as software systems become more and more complex, software programmers needs to know a variety of information and knowledge in various areas. So the Programmers / Company must store knowledge and manage it for reuse. Software reuse is the process of creating software systems from existing software rather than building them from scratch. They described that the software reuses possibilities and measures how much code can be modified from the existing software? If so any problem occurs to the productivity and the comparison of reusable types along with their properties. Finally the reusable software and its cost also discussed.

Shiva et al, in paper [2] discussed that it has been more than three decades since the idea of software reuse was proposed. Many success stories have been told, yet it is believed that software reuse is still in the development phase and has not reached its full potential. How far are we with software reuse research and practice? They proposed the paper to answer this question

Bouchaib et al, in paper [3] described that Testing is an efficient mean for assuring the quality of software. Nowadays, Graphical User Interfaces (GUIs) make up a big part of applications being developed. Within the scope of regression testing, some test cases from the original GUI are usable and others are unusable. The paper presented an algorithm that drops the unusable test cases and

creates new test cases based on the main differences between the two GUIs, which are represented as uncovered edges. Furthermore, the algorithm creates a new test suite for the modified version by combining the usable test cases and the new created test cases.

Sarbjee Singh et al, in paper [4] discussed that Reusability is the likelihood a segment of source code that can be used again to add new functionalities with slight or no modification. Reusable modules and classes reduce implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required. Subroutines or functions are the simplest form of reuse. A chunk of code is regularly organized using modules or namespaces into layers. Proponents claim that objects and software components offer a more advanced form of reusability, although it has been tough to objectively measure and define levels or scores of reusability. Reusability implies some explicit management of build, packaging, distribution, installation, configuration, deployment, maintenance and upgrade issues. If these issues are not considered, software may appear to be reusable from design point of view, but will not be reused in practice.

The paper presented an empirical study of the software reuse activity by expert designers in the context of object-oriented design. The study focused on the three following aspects of reuse : (1) the interaction between some design processes, e.g. constructing a problem representation, searching for and evaluating solutions, and reuse processes, i.e. retrieving and using previous solutions, (2) the mental processes involved in reuse, e.g. example-based retrieval or bottom-up versus top-down expanding of the solution, and (3) the mental representations constructed throughout the reuse activity, e.g. dynamic versus static representations.

Smith et al, in paper [5] stated that the developers and the re-users of software can be readily observed the value of the software. The reuse of existing software helps both developers and users. For the purpose of software preparation software developers needs the existing software. The real time users get advantage from reuse the existing software. Similarly, prospective users of software need support when assessing software for potential reuse.

To evaluate software and related assets for potential reuse, the software developers and software adopters share a common need of capabilities. As software systems become more and more complex, software programmers need to know a variety of information and knowledge in various areas. "Information is wealth", i.e., the knowledge gathered during the development stage can be a valuable asset for a developer as well as the software company. During the software development process, the management and maintenance of knowledge creation is necessary thing. Then only that knowledge is integrated to develop the innovative concept from the older one. So the company must store and manage it for reuse [6].

Terry et al and Kyo Kang et al, in paper [7, 8] discussed that Software metric is a quantitative indicator of an attribute of a software product or process. The relationships among several metrics are specified by metric models. Numerous reuses related metric models have been discussed in literature.

White Howard [9] used a methodology which includes the use of test suite capture data from a capture/replay testing tool. Based on the produced data, White could characterize a test suite for a provided Graphical User Interface using a call graph. This later is mainly based on scores that represent frequent paths selected in diverse test cases which are principally the critical paths. Therefore, these critical paths become indispensable for selecting which unit tests to execute. Kepple, in paper [10] considered the issue of dipping the number of regression test cases to be executed. Their technique is to inspect the various modifications made to numerous parts of an application. If the modifications are within source code that is actually being run by a specific test in a regression test suite, then that specific test should be re-executed. Otherwise, it may be ignored, as that would lead to a very important conclusion which is a safe state without new code being added.

3. PROPOSED METHODOLOGY

3.1 Proposed Method

The aim of the paper is to propose a methodology to develop a software component in order to implement the new emerging technology *Software Reusability*. The proposed methodology provides an efficient solution for how to reuse the software; what the software developer can do to

make the software reusable and how the new software can be reusable in the existing software.

Software is a computer programs and associated documentation such as requirements, design models and user manuals. Software products may be developed for a particular customer or may be developed for a general market. Software Engineering is concerned with theories, methods and tools for professional software development. In other words, Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organized approach to work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

3.1.1 Different Phases in Software Development

To develop software, there involves different phases such as:

- ✚ Analysis Phase
- ✚ Requirement Phase
- ✚ Design Phase
- ✚ Build / Implementation Phase
- ✚ Testing Phase
- ✚ Deployment Phase
- ✚ Maintenance Phase

The brief description of the phases is given below:

- **Analysis:** Some kind of understanding of a problem or situation. This can be obtained by discussing with the client.
- **Requirement:** Based upon the analyzation, what are all the necessary things to be needed to make the solution for the problem. For example, what tool to be used to solve the problem, and so on.
- **Design:** Creation of a solution for the analyzed problem based on the requirements.
- **Build / Implementation:** Write code to implement the design to meet the requirement.
- **Testing:** Test the efficiency and correctness of the developed software.
- **Deployment:** Release the developed software to the client.
- **Maintenance:** Maintain the software by modifying the software as per the client requirements.



Software Reusability is an emerging technology to develop new software from the existing one. This will reduce our manual power, development time and cost. In software engineering, Code Reusability is familiarly known. But, the reusability technology can be implemented in any of the phases listed above.

The software can be developed by following all these phases. The first thing that we can do is to analysis whether the task of the software developer ends with the deployment of the software or still continued to the maintenance phase. If it is continued to maintenance phase, then the project is said to **Maintenance Project** or **Enhancement Project**. In this kind of projects, the programmer has to do minor modifications as per the requirement of the client. To do this, the programmer has to work from Analysis Phase to Deployment Phase.

3.1.1.1. Analysis Phase: The Analysis Phase is categorized into two such as:

- ✓ Initial Analysis
- ✓ Post Golive Analysis

The Post Golive is otherwise called as Deployment or Delivery.

Table 1: Comparison of Initial Analysis Vs Post Golive Analysis

Initial Analysis	Post Golive Analysis
Initial analysis is the analysis that is to be carried out at the initial stage of the project.	Post Golive analysis is carried out by analyzing what are the modification may be occurred after the project delivery.
This can be obtained by discussing with the client about the client needs for the project.	This can be obtained by the analysis team member without the knowledge of the client.
Much Time Consumption and more manual power	Reduced and limited Time. Less Manual Power.

3.1.1.2 Requirement Phase: The requirement phase also has been categorized into 2 phases like the Analysis phase such as:

- ✓ Initial Requirement
- ✓ Post Golive Requirement

Table 2: Comparison of Initial Requirement Vs Post Golive Requirement

Initial Requirement	Post Golive Requirement
Initial requirement is to be carried out at the initial stage of the project.	Post Golive requirement is carried out by analyzing what are the requirements may be occurred to modify the project after the project delivery.
This can be done from the analysis report which was obtained from the client.	This can be obtained by the requirement team member without the knowledge of the client.
Much time consumption and required more manual power	Less time consumption and with reduced manual power.

3.1.1.3 Design Phase: The design phase can be categorized as:

- ✓ Initial Design
- ✓ Post Golive Design

Table 3: Comparison of Initial Design Vs Post Golive Design

Initial Design	Post Golive Design
This is to be done at the initial stage of the project based on the analysis and requirements report.	Post Golive design is done by the design team analyzing what are the modification in the design phase may be occurred after the project delivery and what are the tools to be required to do that.

3.1.1.4 Build Phase:The Build phase can be categorized as:

- ✓ Initial Build
- ✓ Post Golive Build

Table 4: Comparison of Initial Build Vs Post Golive Build

Initial Build	Post Golive Build
This is to be done at the initial stage of the project by writing the code for the project to implement the design.	Post Golive Build is done by the Implementation team analyzing what are the correction may be occurred in the coding after the project delivery and how to do those changes.

3.1.1.5 Testing Phase:The Testing phase can be categorized as:

- ✓ Initial Testing
- ✓ Post Golive Testing

Table 5: Comparison of Initial Testing Vs Post Golive Testing

Initial Testing	Post Golive Testing
This is to be done at the initial stage of the project after completing the build phase.	Post Golive Testing is done by the testing team analyzing what are the new testing may be carried out upon the project deployment and how those testing may be carried out along with the cost factors.

Thus these Initial and Post Golive categories of all the phases provides a set of documents describing the list of activities to be occurred in the project in all the phases at the initial stage as well as at the post golive stage. This document is known as **Impact Analysis Document (IA Document)**.

3.1.2 Impact Analysis Document

The Impact Analysis Document helps the Software Engineers to develop efficient software. The Programmers are divided into 4 teams:

- ❖ Business Analysis Team
- ❖ Development Team
- ❖ Testing Team
- ❖ Business Intelligence Team

At first, these team members analyze the problem and create the Impact Analysis Document for future use. The document contains information as Templates. The template contains all the relevant details about the project that can be created by these team members. A sample is given below:

Table 6: Sample IA Document

Team	Task
Business Analysis Team	<ul style="list-style-type: none"> ▪ Get the requirements from the client ▪ How many days to draw the flow of outline of the project? ▪ How many days required to complete the projects? ▪ What is the configuration required for the project? ▪ What front-end will be used in this project? ▪ Is there any possibility to change the front-end of the project? ▪ What database will be used to store the data? ▪ Are there any new additional fields to be included in the database?

Development Team	<ul style="list-style-type: none"> ▪ What are the local and global variables used to implement the project? ▪ How many days required to write the code of the project? ▪ What are the tables used in the implementation part? ▪ Is there any need to create Procedure for the project? ▪ Is there any new column to be added in the table for the project? ▪ From where the project begins?
Testing Team	<ul style="list-style-type: none"> ▪ What type of testing required for the project? ▪ System Testing – Yes / No ▪ Component Testing – Yes / No ▪ Regression Testing – Yes / No ▪ Integration Testing – Yes / No ▪ User Acceptance Testing – Yes / No
Business Intelligence Team	<ul style="list-style-type: none"> ▪ Is there any new report generated for the project or the existing report modify? ▪ Is there any global / local changes needed? ▪ How many working days did the project require? ▪ How many developers did the project use? ▪ When did the project start? ▪ When did the project end?

By filling the answer for the above question, the IA Document can be generated and converted into Template, which will be used in future. Based upon the request, the programmer fills the template and reuses the software.

3.1.3 Requirement Traceability Matrix (RTM)

The Requirement Traceability Matrix (RTM) contains the storage location about the project which helps the programmer to identify and locate the data that can be used in their project. This RTM contains the details about the project along with all the phases. The sample RTM is shown below in Table-7.

The specification for the design, build and testing can be maintained in a specified location which would be linked in this RTM. The sample specification of design, build and testing are given below:

C:\login\deslog:

Draw 2 labels

Draw 2 textfields

Draw a Button

Label1.Text = "Username"

Label2.Text = "Password"

Textbox2.char = "*"

Button1.Text = "Login"

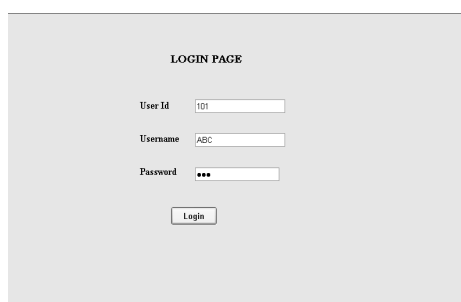


Figure 2: Sample Form Design

C:\login\code\log:

Declare 2 variables as string, s1,s2

Get the values from the textbox.

Click the button

C:\login\UT\log:

The testing data is shown in the Table-8.

Thus by using this RTM, the programmer can reuse this login page to any another project. The reuse can be done by navigate through the link given for the project.

Software Reusability is the process of reusing the existing thing to create the new thing. The thing may not only be the code of the project, but also the design document of the project. In other words, Software Reusability denotes not only for code reusability, but also for design document reusability. This will be implemented in our paper, by developing a new initial approach called **IA Document**. This document contains all the necessary information about the software. When the new project comes, the programmer can first analysis this *IA Document* to verify whether the IA Document contains the Software that is related with the new project. If so, then the programmer can reuse the existing software to develop the new one. If there is any change, the programmer can fix that relevant column only, without modifying the whole software. Thus it saves the time and cost to develop the new project. Also, the manual power can be reduced. The major advantage of this IA

Document is that the client can easily understand about the software through this document.

The major success of the software relates with the client satisfaction, which will deal on completing the software by implementing all the client requirements without any problem. (i.e), the Software will meet the requirements 100%. To meet this 100% requirement, we propose the new approach called **RTM (Requirement Traceability Matrix)**. After completing the project, the RTM can be prepared by the whole team. Tracing this RTM by the whole team, the project should be delivered with 100% performance.

Thus, our paper proposes a methodology to develop a **Reusable Asset** also known as **Software Component** by implementing two major approaches such as,

- ✓ **Impact Analysis Document (IA Doc)** and
- ✓ **Requirement Traceability Matrix (RTM)**

for software reusability. This will helps the programmer to handle several projects easily. Through this, the programmer can reuse the entire software cycle since the IA Document handle the software from Initial Phase to Build Phase whereas the RTM handles the Deployment Phase.

3.2 Algorithm

Begin

Get the Requirements from the Client

Organize the Requirements

Analysis in Impact Analysis Document (IA Document)

If the Requirement match with the IA Document then

Project type = "Existing"

Check the Requirement Traceability Matrix (RTM)

Match the requirement with the RTM

If match = "success" then

If data = "present" then

Use the link and navigate

Fill the Value

Else

Skip it

End if

Else

Fill the new value and execute

End if

Else

Project type = "New"

Create new IA Document and RTM

End if

End

3.2.1 Algorithm Explanation

The first step is to get the requirements from the client and organize it. Then analysis the IA Document to verify if there is any document matches with the new requirements. If so, the project reuses the existing software by using the RTM. With the help of RTM, the programmer fills the value for the new project. Using the data in the RTM, the programmer executes the new software easily. Thus the software reusability can be efficiently done by using the IA Document and RTM.

4. EXPERIMENTAL DATA

The proposed methodology is very efficient for software reusability by developing a reusable asset using the Impact Analysis Document and Requirement Traceability Matrix. To test the efficiency of the methodology, various experimental setups are constructed and the result is analyzed. The experiment is made among the software engineers to develop 5 projects. Of those projects, 3 projects are of relevant types, whereas remaining 2 are entirely different from each other. The time period, manual work, efficiency is all calculated for these 5 projects and then compare the results to identify whether our proposed work of software reusability performs well. The comparison result is shown in Table-9.

From this comparison data, the projects 2 and 3 reuse the project 1 and so the staffs, effort and latent defects are reduced; whereas the projects 4 and 5 are independent and so these values are increased. Thus our proposed methodology performs well. We also compare the performance of our methodology as:

Table 10: Performance of our Methodology

Documentation and Ready Status	Always
Balancing the Development Cost	Yes
Modification & Open Standard Reference	Often
Reuse Design & Testing	Always
Updating and Improvement	Often

This table shows the characteristics of the reusable software and our methodology meet all these characteristics and proves that our proposed methodology is the better one.

5. CONCLUSIONS

The success of the software reusability hinges on the disciplined implementation of the proposed model. The effectiveness can be enhanced by placing the reusable components in the suitable and global repositories which offers suitable incentives to maximize and institutionalize re-use. Reuse processes and procedures must be incorporated into the existing software development process. Repositories of software assets must be created and maintained and must be designed for reusability. Through our work, the software quality and reliability can be increased with less time and cost. Thus we conclude that Software reusability is an emerging technology, which save the production cost, improving the innovative technology from the existing one. This reuse research has been ongoing since the late 1960s. Much has been accomplished, but there is still much to do to provide better results via the reuse are completely achieved. Though most organizations reuse components to save the time and cost, reuse is never risk free.

This paper performs well and it satisfies the aim of the paper by developing a reusable asset using *Impact Analysis Document* and *Requirement Traceability Matrix*. Our paper is being important, since our methodology can be applicable for all kinds of projects. It can be carried out by evaluating the project from analysis phase to deployment phase.

Much has been accomplished, but there is still much to do before the vision of better system building via reusing the software. A possible future work will be to use these findings that we found in our paper and continue the process to enhance the project by modifying any of the phases from analysis to deployment rather than using all the phases, depending upon the purpose of the project.

REFERENCES:

- [1] G.N.K. Suresh Babu and Dr.S.K. Srivatsa, "Analysis and Measures of Software Reusability", *International Journal of Reviews in Computing* E-ISSN: 2076-3336, 2009.
- [2] Sajjan G. Shiva and Lubna Abou Shala, "Software Reuse: Research and Practice", *International Conference on Information Technology* 0-7695-2776-0/07, 2007.
- [3] Bouchaib Falah, Rahima Nouasse, Yassine Laghid, "GUI Regression Test Selection Based on Event Interaction Graph Strategy", *IJCSET* | Vol 3, Issue 3, 76-79, March 2013.



- [4] Sarbjeet Singh, Sukhvinder Singh, Gurpreet Singh, "Reusability of the Software", *International Journal of Computer Applications* (0975-8887), Volume 7-No.14, October 2010.
- [5] Smith and Williams 2002] C. U. Smith and L. G. Williams, "Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software", Boston, MA, Addison-Wesley, 2002.
- [6] Software Engineering, vol SE- 12 no. 1 1994. Gert B (1988) *Morality*, Oxford University Press.
- [7] W. Frakes, and C. Terry, "Software Reuse: Metrics and Models", *ACM Computing Surveys*, vol. 28, no 2, 1996, pp. 415-435.
- [8] W. Frakes, and Kyo Kang, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, vol. 31, no. 7, 2005, pp 529-536.
- [9] L. White. Regression testing of GUI event interactions. *In Proceedings of the International Conference on Software Maintenance*, pages 350–358, Washington, Nov.4–8 1996.
- [10] KEPPLE, L. R, "The black art of GUI testing", *Dr. Dobbs's J. Softw.Tools* 19, 2 (Feb.), 40, 1994.



Table 7: Sample RTM

ID	Requirement	Design	Build	Unit Testing	System Testing	Integration Test	UAT
1	Validate the Login Page	Link:c:\logi n\deslog	Link:c:\logi n\codelog	Link:c:\logi n\UTlog\col 6	Link:c:\login\ UTlog\col7	Link:c:\login\ UTlog\col8	Link:c:\ login\ UTlog\ col9
2							
3							
4							
5							
...							

Table-8: Testing Report

Project Name: <pjct_name>					
Test Case Template					
Test Case ID: <id>					
Test Priority : <low/high/medium>					
Test Title : Verify the Login Details					
Unit Testing : <yes / no>					
System Testing : <yes / no>					
Integration Testing : <yes / no>					
UAT : <yes / no>					
Description : Test the login value to verify the user					
Test Designed by : <name>					
Test Designed date : <date>					
Test Executed by : <name>					
Test Execution date : <date>					
Pre-Conditions: User has valid details					
Post-Conditions: User is validated with the database and successfully login to the project					
Step	Test Steps	Test Data	Expected Results	Status	Note
1	Navigate to login page	C:\log.html	User should be move on to login page	Pass	
2	Provide Username	Admin	Admin	Pass	
3	Provide Password	Admin	Admin123	Fail	
4	Click 'login' button	Click the button		Pass	



Table-9: Comparison Result

Project	Reusable Percentage	Staffs	Effort (in Months)	Schedule (Hr/Day)	Defect Potential	Removal Efficiency	Latent Defects
1	0.00%	7	10	12:00	953	89.00%	105
2	100%	1	4	12:00	2	99.00%	0.02
3	100%	1	2	12:00	1	99.00%	0.01
4	0.00%	5	9	12:00	897	85.00%	135
5	0.00%	9	15	12:00	978	80.00%	196

Note: Latent Defects = Defect Potential – (Removal Efficiency * Defect Potential)